Pangomonium 2.0

The Ultimate Text & Graphics Engine for Hollywood

Andreas Falkenhahn

Table of Contents

1 General information

1.1 Introduction

Pangomonium is a plugin for Hollywood that features advanced rendering engines both for text and graphics. Pangomonium's text rendering engine features state-of-the-art layouting and makes it possible to draw text in almost every language of the world. It supports complex layouts like the right-to-left or bidirectional ones found in the Arabic and Hebrew languages or vertical layouts in columns like in Japanese. Pangomonium can also handle complex-text languages like Hindi and fonts that contain colored glyphs (emojis) are supported as well. By giving you full access to the Pango API the plugin also enables you to customize all stages of the text rendering process.

Furthermore, Pangomonium also comes with a loader for the popular SVG vector image format. As soon as Pangomonium is installed, Hollywood will "automagically" be able to load and draw SVG images. SVG images will be loaded as true vector images by Pangomonium which means that you can scale and transform them as you wish without any losses in quality. They will be perfectly crisp in any resolution.

On top of that, Pangomonium also provides Hollywood wrappers for almost all functions of the popular Cairo graphics engine, allowing you to access advanced vector graphics drawing features directly from Hollywood scripts. Using the Cairo API via Pangomonium has the advantage that you have fine-tuned control over everything when drawing vector graphics.

There are two ways of using Pangomonium: There is a high-level interface that can directly hook itself into Hollywood's text and vector graphics library, enhancing it with features provided by Pangomonium, e.g. drawing colored emojis or text with complex layouts such as Arabic or Japanese. This is the most convenient way of using Pangomonium because you don't have to use the Pango and Cairo functions directly here but you can just use established Hollywood functions.

Another way of using Pangomonium is the low-level interface: This interface allows you to access the Pango and Cairo APIs directly from Hollywood scripts. This is extremely powerful because it allows you to access hundreds of different text and graphics rendering features, making it possible to fine-tune Pangomonium to your specific needs. Pangomonium contains over 500 commands to make all your text and vector graphics rendering dreams come true!

Finally, Pangomonium comes with extensive documentation in various formats like PDF, HTML, AmigaGuide, and CHM that contains detailed descriptions of all functions and methods offered by the plugin.

All of this makes Pangomonium the ultimate text and graphics rendering engine for Hollywood that contains everything you need to draw text in any language spoken on the planet.

1.2 Terms and conditions

The program is provided "as-is" and the author cannot be made responsible of any possible harm done by it. You are using this program absolutely at your own risk. No warranties are implied or given by the author.

This plugin may be freely distributed as long as the following three conditions are met:

1. No modifications must be made to the plugin.
2. It is not allowed to sell this plugin.
3. If you want to put this plugin on a coverdisc, you need to ask for permission first.

This software uses Pango which is released under the terms of the GNU Lesser General Public License. See Section A.1 [LGPL license], page 341, for details.

This software uses GNU FriBidi which is released under the terms of the GNU Lesser General Public License. See Section A.1 [LGPL license], page 341, for details.

This software uses Cairo which is released under the terms of the GNU Lesser General Public License. See Section A.1 [LGPL license], page 341, for details.

This software uses GLib which is released under the terms of the GNU Lesser General Public License. See Section A.1 [LGPL license], page 341, for details.

This software uses Librsvg which is released under the terms of the GNU Lesser General Public License. See Section A.1 [LGPL license], page 341, for details.

This software uses the Croco library which is released under the terms of the GNU Lesser General Public License. See Section A.1 [LGPL license], page 341, for details.

This software uses HarfBuzz. See Section A.2 [HarfBuzz license], page 348, for details.

This software uses Expat which is Copyright (c) 1998-2000 Thai Open Source Software Center Ltd and Clark Cooper and Copyright (c) 2001-2022 Expat maintainers. See Section A.3 [Expat license], page 348, for details.

This software uses libxml2 which is Copyright (C) 1998-2012 Daniel Veillard. See Section A.6 [Libxml2 license], page 350, for details.

This software uses Fontconfig. See Section A.4 [Fontconfig license], page 349, for details.

This software uses the pixman library. See Section A.5 [Pixman license], page 349, for details.

This software uses libpng by the PNG Development Group and zlib by Jean-loup Gailly and Mark Adler.

Portions of this software are copyright (C) 2023 The FreeType Project (www.freetype.org). All rights reserved.

All trademarks are the property of their respective owners.

DISCLAIMER: THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDER AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1.3 Requirements

- Hollywood 10.0 or better
- Windows: requires Windows Vista or better
- macOS: requires at least 10.5 (Leopard) for PowerPC, 10.9 (Mavericks) for x64 and 11.0 (Big Sur) for arm64
- AmigaOS 3: your filesystem must be able to handle filenames longer than the usual 31 characters which are the limit on classic FFS; so you mustn't use classic FFS with Pangomonium; also, when using fonts that contain emojis or CJK characters you'll also need lots of memory because those fonts are often than 10 to 20 megabytes (or even more) in size and Pangomonium loads each font you use into a contiguous memory block so you'll need lots of memory and it mustn't be fragmented so that large blocks can still be allocated

1.4 Installation

Installing Pangomonium is straightforward and simple: Just copy the file `pangomonium.hwp` for the platform of your choice to Hollywood's plugins directory. On all systems except on AmigaOS and compatibles, plugins must be stored in a directory named `Plugins` that is in the same directory as the main Hollywood program. On AmigaOS and compatible systems, plugins must be installed to `LIBS:Hollywood` instead. On macOS, the `Plugins` directory must be inside the `Resources` directory of the application bundle, i.e. inside the `HollywoodInterpreter.app/Contents/Resources` directory. Note that `HollywoodInterpreter.app` is stored inside the `Hollywood.app` application bundle itself, namely in `Hollywood.app/Contents/Resources`.

Afterwards merge the contents of the `Examples` folder with the `Examples` folder that is part of your Hollywood installation. All Pangomonium examples will then appear in Hollywood's GUI and you can launch and view them conveniently from the Hollywood GUI or IDE.

On Windows you should also copy the file `Pangomonium.chm` to the `Docs` directory of your Hollywood installation. Then you will be able to get online help by pressing F1 when the cursor is over a Pangomonium function in the Hollywood IDE.

On Linux and macOS copy the `Pangomonium` directory that is inside the `Docs` directory of the Pangomonium distribution archive to the `Docs` directory of your Hollywood installation. Note that on macOS the `Docs` directory is within the `Hollywood.app` application bundle, i.e. in `Hollywood.app/Contents/Resources/Docs`.

2 About Pangomonium

2.1 Credits

Pangomonium was written by Andreas Falkenhahn, based on the Pango text engine and other components. See Section 1.2 [Pangomonium license], page 1, for details.

If you need to contact me, you can either send an e-mail to `andreas@airsoftsoftwair.de` or use the contact form on `http://www.hollywood-mal.com`.

2.2 Frequently asked questions

This section covers some frequently asked questions. Please read them first before asking on the forum because your problem might have been covered here.

Q: Is there a Hollywood forum where I can get in touch with other users?

A: Yes, please check out the "Community" section of the official Hollywood Portal online at `http://www.hollywood-mal.com`.

Q: Where can I ask for help?

A: There's an active forum at `http://forums.hollywood-mal.com`. You're welcome to join it and ask your question there.

Q: I have found a bug.

A: Please post about it in the "Bugs" section of the forum.

2.3 Future

Here are some things that are on my to do list:

- add more examples

Don't hesitate to contact me if Pangomonium lacks a certain feature that is important for your project.

2.4 History

Please see the file `history.txt` for a complete change log of Pangomonium.

3 Using Pangomonium

3.1 Overview

There are two different ways of using Pangomonium: You can either access the Pango and Cairo APIs directly through a low-level interface or you can use Pangomonium's high-level interface which maps some of Pangomonium's features to standard Hollywood functions.

Using the high-level interface is really easy and extends Hollywood functions like `SetFont()` and `TextOut()` to operate through Pangomonium, enabling them to support advanced text layouts like right-to-left or colored glyphs (emojis). If you just want to draw some text and you don't need any fine-tuned control over the text rendering process, the high-level interface is the way to go for you.

Besides the high-level interface that extends Hollywood's text library, Pangomonium also has a second high-level interface. In contrast to the first one, the second high-level interface replaces Hollywood's vector graphics library and can be activated by calling Hollywood's `SetVectorEngine()` function and passing "pangomonium" as the argument.

The low-level interface, i.e. accessing Pango and Cairo's API directly, is useful if you need more fine-tuned control over the text rendering. The low-level interface allows you to configure all kinds of options in Pangomonium and makes it possible to access all of Pango and Cairo's advanced features, allowing you to meticulously take control over everything.

3.2 Using the high-level interface

Using Pangomonium's high-level interface is really easy. It extends standard Hollywood text commands like `SetFont()` and `TextOut()` to support advanced text rendering features like complex layouts and colored glyphs like emojis. To draw text using Pangomonium through normal Hollywood commands, just make sure you open the font using Pangomonium by passing the `Loader` tag to `SetFont()`. Once you have opened a font through Pangomonium, all Hollywood commands that draw text will automatically do so through Pangomonium as well, e.g.:

```
@REQUIRE "pangomonium"

SetFont("Noto Sans CJK HK Regular", 72, {Loader = "pangomonium"})
SetFontStyle(#ANTIALIAS)

; "What is your name?" in Japanese
Local t = {21517, 21069, 12399, 12394, 12435, 12391, 12377, 12363}
For Local k = 0 To ListItems(t) - 1 Do text$ = text$ .. Chr(t[k])

TextOut(#CENTER, #CENTER, text$)
```

The code above will set the font to Noto Sans CJK HK Regular size 72 and draw some Japanese text. By passing "pangomonium" to the `Loader` tag you tell `SetFont()` to open the font through Pangomonium instead of through Hollywood's inbuilt font engine or through a font engine provided by the host operating system.

Another advantage of using the high-level interface is the fact that Pangomonium supports true vector-based text transformation without any quality loss. Hollywood's inbuilt font

engines only support lossless scaling of vector text. As soon as text is rotated, a lossy algorithm is used so drawing rotated text using Pangomonium will give you a better quality.

Furthermore, Pangomonium allows you to use Pango's markup language which can do some advanced text formatting that Hollywood's inbuilt text engine can't do. For example, Pango's markup language allows you to use different font sizes in a single text object. So if you need more powerful formatting capabilities than supported by Hollywood's inbuilt text renderer, you might also want to use Pangomonium instead. See playout:SetMarkup() for more information on Pango's markup language. Also note that Pango's markup language isn't enabled by default in Pangomonium for compatibility reasons. If you want to use it, you need enable it explicitly by setting the `Markup` tag to `True` (see below).

Note that in the code above we explicitly tell `SetFont()` to use Pangomonium to open the font by passing `"pangomonium"` in the `Loader` tag. Alternatively, you can also globally enable Pangomonium for all Hollywood commands dealing with fonts by simply installing Pangomonium's font adapter. This can be done by setting the `InstallAdapter` tag to `True` when `@REQUIREing` Pangomonium, e.g. like this:

```
@REQUIRE "pangomonium", {InstallAdapter = True}
```

If you globally enable Pangomonium, you don't have to use the `Loader` tag with `SetFont()` or other Hollywood commands that deal with fonts any longer because if Pangomonium is globally enabled, commands like `SetFont()` will automatically ask Pangomonium for every font that is to be opened whether or not the plugin would like to handle this font. Keep in mind, though, that by default Pangomonium will never fail on font opening so if you globally activate it like above, it will take over the handling of each and every font you open, which might not be what you want. You can modify this behaviour by setting the `NoFail` user tag to `False`, in which case Pangomonium will fail if a font is not available. If you want Pangomonium to handle all fonts in your script, however, globally enabling it using `InstallAdapter` as shown above can be a good idea.

When using Pangomonium you can also pass some additional arguments to `SetFont()`, `OpenFont()` or the `@FONT` preprocessor by using Hollywood's user tags. The following additional arguments are recognized by Pangomonium:

`NoFail` By default, Pangomonium will never fail to open a font. If the requested font does not exist, an appropriate substitute is used. By default, Pangomonium will always open a font no matter what name you pass to it. If you don't want that, set this tag to `False`. Defaults to `True`.

`Markup` If you set this to `True`, Pangomonium will allow you to use the Pango markup language in the text you pass to Hollywood calls like `TextOut()`. By default, Pangomonium only supports Hollywood's text formatting codes. If you want it to use Pango's markup language instead of Hollywood's text formatting codes, set this tag to `True`. See playout:SetMarkup() for information on Pango's markup language. Defaults to `False`.

`Monochrome`
Set this tag to `True` if you want Pangomonium to operate in monochrome mode. This will reduce memory consumption by 75% because Pangomonium only needs to allocate one pixel channel instead of four but you will only be able to use single-colored text in monochrome mode so things like color emojis won't be drawn correctly. Defaults to `False`.

`NoFT2` By default, Pangomonium uses Pango's FreeType2-based text renderer on all platforms. If you don't want that, set this tag to `True`. In that case, Pangomonium will use the host operating system's default text renderer instead (e.g. DirectWrite on Windows, Core Text on macOS etc.) Note that on AmigaOS and compatibles the only available renderer is FreeType2 so you mustn't set this tag to `True` on AmigaOS. Defaults to `False`.

Here's how you can pass user tags to Pangomonium:

```
@REQUIRE "pangomonium"
SetFont("Arial", 72, {Loader = "pangomonium", UserTags = {Markup = True}})
SetFontStyle(#ANTIALIAS)
TextOut(#CENTER, #CENTER, "<span bgcolor=\"red\">Hello</span>")
```

The code above demonstrates how to enable Pango's markup language in Pangomonium's high-level interface.

3.3 Using the vectorgraphics interface

Besides the high-level interface that extends Hollywood's text functions, Pangomonium also has a second high-level interface that can be used to replace Hollywood's vector graphics library. You can use Hollywood's `SetVectorEngine()` function to make Pangomonium the default renderer for vector graphics. This means that all vector graphics will be drawn by Pangomonium instead of Hollywood, so Hollywood functions like `DrawPath()` and all other functions from the vector graphics library will be handled by Pangomonium.

To make Pangomonium the default renderer for vector graphics, just do the following:

```
SetVectorEngine("pangomonium")
```

3.4 Loading SVG images

Pangomonium can also load and draw SVG images. As soon as the plugin is installed, Hollywood will 'automagically' be able to open SVG images. So when you use `LoadBrush()`, for example, on an SVG image, Pangomonium will automatically load it. Note that since there is another Hollywood plugin that can load SVG images, it is recommended that you use the `Loader` tag to specify which plugin should be used to load the SVG image, e.g.

```
; this code will make Pangomonium load the image
LoadBrush(1, "test.svg", {Loader = "pangomonium"})

; this code will make the svgimage.hwp plugin load the image
LoadBrush(1, "test.svg", {Loader = "svgimage"})
```

If you don't specify the `Loader` tag and you have two plugins that can load SVG images installed, the plugin that Hollywood has loaded first will be the one that will load your SVG image. But since the order Hollywood plugins are loaded is dependent on the file system you can't really rely on anything so it's better to explicitly use the `Loader` tag to tell Hollywood which plugin to use to load the SVG image. To see which plugin loaded your SVG image, you can use the `#ATTRLOADER` attribute like this:

```
DebugPrint(GetAttribute(#BRUSH, 1, #ATTRLOADER))
```

When it loads an SVG image, Pangomonium will always create a vector brush for you. This means that you can transform it using commands like `ScaleBrush()`, `RotateBrush()`, and

`TransformBrush()` without any losses in quality. In case layers are enabled and you display a vector brush using `DisplayBrush()`, the layer graphics can also be transformed without any quality loss. The same is true for BGPics using vector graphics.

Please note that Pangomonium will also always create images with alpha channel by default. If you don't want that, you have to manually delete the alpha channel using `DeleteAlphaChannel()` after loading the image. In case you want to use an SVG image as a BGPic, you can also work around this problem by defining a background for this BGPic, e.g. like this:

```
@BGPIC 1, "test.svg", {FillStyle = #FILLCOLOR, FillColor = #WHITE,
                       Loader = "pangomonium"}
```

3.5 Using the low-level interface

Using Pangomonium's low-level interface is more difficult than using the high-level interface because it allows you to access Pango and Cairo APIs directly. This means that you should first make yourself familiar with those APIs so that you know how they are designed and how they can serve your purposes.

On top of the core Pango and Cairo APIs, Pangomonium also offers some special functions that can be used to bridge Pango/Cairo calls and Hollywood functions. One of the key functions that serves as such a bridge is csurface:ToBrush() which allows you to convert a Cairo surface to a Hollywood brush. Conversely, it is also possible to create Cairo surfaces from Hollywood brushes by using the cairo.ImageSurfaceFromBrush() function.

By using these bridging functions, you can use Pango/Cairo APIs directly and then convert the result to a Hollywood brush. For example, drawing text using Pango typically involves the following steps:

1. Create a Pango fontmap
2. Create a Pango context from the fontmap
3. Create a Pango layout from the context
4. Create a Pango font description and assign it to the layout
5. Load a font into the fontmap
6. Set text that is to be rendered
7. Create a Cairo surface
8. Create a Cairo context from the surface
9. Show the Pango layout on the Cairo context
10. Convert the Cairo surface to a Hollywood brush

As you can see, it's quite some work to draw text with Pangomonium when using the Pango/Cairo APIs directly. It's much easier to do it with the high-level interface but only the low-level interface allows you to access all of Pango's and Cairo's features.

Here is what the steps described above look like in actual code:

```
@REQUIRE "pangomonium"
@DISPLAY {Color = #WHITE}

fontmap = pango.FontMap(#CAIRO_FONT_TYPE_FT)
```

```
context = fontmap:CreateContext()
layout = pango.Layout(context)
fontdesc = pango.FontDescription("DejaVu Sans 72")
layout:SetFontDescription(fontdesc)
fontmap:LoadFont(context, fontdesc)
layout:SetText("Hello World")
img = cairo.ImageSurface(#CAIRO_FORMAT_ARGB32, 640, 480)
cr = cairo.Context(img)
cr:ShowLayout(layout)
img:ToBrush(1)
DisplayBrush(1, 0, 0)
```

When using the low-level interface you also need to think about freeing objects when they are no longer in use. This can be done by calling the `Free()` methods of the individual objects, e.g. pcontext:Free() to free a Pango context. Alternatively, you can also set the object to `Nil`. This signals Hollywood's garbage collector that the object is no longer in use and can be freed. However, keep in mind that garbage collection will only occur if there are really no more references to the object anywhere. Thus, it might be safer to explicitly call the `Free()` methods instead of setting objects to `Nil`.

If you put the code from the example above in a function and use local variables only, however, you can get away without explicitly calling `Free()` on all objects because local variables will automatically be garbage-collected once they go out of scope. Consider the following code, which is the same as above with the only difference that the code is now wrapped inside a function:

```
Function p_DrawText(t$, brush)
   Local fontmap = pango.FontMap()
   Local context = fontmap:CreateContext()
   Local layout = pango.Layout(context)
   Local fontdesc = pango.FontDescription("DejaVu Sans 72")
   layout:SetFontDescription(fontdesc)
   fontmap:LoadFont(context, fontdesc)
   layout:SetText(t$)
   Local img = cairo.ImageSurface(#CAIRO_FORMAT_ARGB32, 640, 480)
   Local cr = cairo.Context(img)
   cr:ShowLayout(layout)
   img:ToBrush(brush)
EndFunction
```

Since all Pango and Cairo objects are stored inside local variables, everything will be marked for garbage collection as soon as the function returns. Thus, in this case it isn't necessary to explicitly free objects by calling `Free()`, even though it might be considered good practice to do so nevertheless.

For more information on the Pango and Cairo APIs, please refer to the following chapters.

4 Cairo functions

4.1 cairo.Context

NAME

cairo.Context – create Cairo context

SYNOPSIS

```
handle = cairo.Context(target)
```

FUNCTION

Creates a new Cairo context with all graphics state parameters set to default values and with `target` as a target surface. The target surface should be constructed with a backend-specific function such as cairo.ImageSurface() (or any other Cairo surface backend constructor).

This function references `target`, so you can immediately call csurface:Free() on it if you don't need to maintain a separate reference to it.

This function returns a newly allocated Cairo context with a reference count of 1. The initial reference count should be released with ccontext:Free() when you are done using the Cairo context. This function never returns `Nil`. If memory cannot be allocated, a special Cairo context object will be returned on which ccontext:Status() returns `#CAIRO_STATUS_NO_MEMORY`. If you attempt to target a surface which does not support writing (such as a Cairo MIME surface) then a `#CAIRO_STATUS_WRITE_ERROR` will be raised. You can use this object normally, but no drawing will be done.

INPUTS

`target` target surface for the context

RESULTS

`handle` a newly allocated Cairo context

4.2 cairo.FontFace

NAME

cairo.FontFace – create font face

SYNOPSIS

```
font = cairo.FontFace(pattern)
```

FUNCTION

Creates a new font face for the FreeType font backend based on a Fontconfig pattern. This font can then be used with ccontext:SetFontFace() or cairo.ScaledFont().

The `pattern` argument can be either a string in the format used by Fontconfig, e.g. `"Ubuntu:style=italic:weight=200"` or it can be a table that can contain the following tags, each describing a single element of a Fontconfig pattern:

`Family` The font family name.

`Style` Font style (passed as a string). Overrides weight and slant.

`Slant` Font slant. This can be one of the following constants:

```
#FC_SLANT_ROMAN
#FC_SLANT_ITALIC
#FC_SLANT_OBLIQUE
```

`Weight` Font weight. This can be one of the following constants:

```
#FC_WEIGHT_THIN
#FC_WEIGHT_EXTRALIGHT
#FC_WEIGHT_ULTRALIGHT
#FC_WEIGHT_LIGHT
#FC_WEIGHT_DEMILIGHT
#FC_WEIGHT_SEMILIGHT
#FC_WEIGHT_BOOK
#FC_WEIGHT_REGULAR
#FC_WEIGHT_NORMAL
#FC_WEIGHT_MEDIUM
#FC_WEIGHT_DEMIBOLD
#FC_WEIGHT_SEMIBOLD
#FC_WEIGHT_BOLD
#FC_WEIGHT_EXTRABOLD
#FC_WEIGHT_ULTRABOLD
#FC_WEIGHT_BLACK
#FC_WEIGHT_HEAVY
#FC_WEIGHT_EXTRABLACK
#FC_WEIGHT_ULTRABLACK
```

`Size` Font size in points.

`Aspect` Stretches glyphs horizontally before hinting.

`PixelSize`
Font size in pixels.

`Spacing` Font spacing. This can be one of the following constants:

```
#FC_PROPORTIONAL
#FC_DUAL
#FC_MONO
#FC_CHARCELL
```

`Width` Font width. This can be one of the following constants:

```
#FC_WIDTH_ULTRACONDENSED
#FC_WIDTH_EXTRACONDENSED
#FC_WIDTH_CONDENSED
#FC_WIDTH_SEMICONDENSED
#FC_WIDTH_NORMAL
#FC_WIDTH_SEMIEXPANDED
#FC_WIDTH_EXPANDED
#FC_WIDTH_EXTRAEXPANDED
#FC_WIDTH_ULTRAEXPANDED
```

`File` The filename holding the font.

`Index` The index of the font within the file.

This function returns a newly created Cairo font face. Free with cfontface:Free() when you are done using it.

INPUTS

`pattern` a Fontconfig pattern, passed either as a string or table

RESULTS

`font` a newly created Cairo font face

4.3 cairo.FontOptions

NAME

cairo.FontOptions – create new font options object

SYNOPSIS

```
handle = cairo.FontOptions()
```

FUNCTION

Allocates a new font options object with all options initialized to default values.

This function returns a newly allocated Cairo font options object. Free with cfontoptions:Free(). This function always returns a valid handle; if memory cannot be allocated, then a special error object is returned where all operations on the object do nothing. You can check for this with cfontoptions:Status().

INPUTS

none

RESULTS

`handle` a newly allocated Cairo font options object

4.4 cairo.Glyphs

NAME

cairo.Glyphs – create new glyphs array

SYNOPSIS

```
handle = cairo.Glyphs(n)
```

FUNCTION

Allocates a new glyphs array that has enough room for storing `n` glyphs. You can then set and get the individual glyph at a specific index by using the cglyphs:Get() and cglyphs:Set() functions. By default, all glyph IDs and offsets in the object will be set to 0.

A font is (in simple terms) a collection of shapes used to draw text. A glyph is one of these shapes. There can be multiple glyphs for a single character (alternates to be used

in different contexts, for example), or a glyph can be a ligature of multiple characters. Cairo doesn't expose any way of converting input text into glyphs, so in order to use the Cairo interfaces that take arrays of glyphs, you must directly access the appropriate underlying font system.

INPUTS

none

RESULTS

`handle` a newly allocated Cairo glyphs array

4.5 cairo.ImageSurface

NAME

cairo.ImageSurface – create image surface

SYNOPSIS

```
handle = cairo.ImageSurface(format, width, height)
```

FUNCTION

Creates an image surface of the specified format and dimensions. Initially the surface contents are set to 0. (Specifically, within each pixel, each color or alpha channel belonging to format will be 0. The contents of bits within a pixel, but not belonging to the given format are undefined).

The `format` argument can be one of the following pixel formats:

`#CAIRO_FORMAT_ARGB32`
: Each pixel is a 32-bit quantity, with alpha in the upper 8 bits, then red, then green, then blue. The 32-bit quantities are stored native-endian. Pre-multiplied alpha is used. (That is, 50% transparent red is $80800000, not $80ff0000.)

`#CAIRO_FORMAT_RGB24`
: Each pixel is a 32-bit quantity, with the upper 8 bits unused. Red, green, and blue are stored in the remaining 24 bits in that order.

`#CAIRO_FORMAT_A8`
: Each pixel is a 8-bit quantity holding an alpha value.

`#CAIRO_FORMAT_A1`
: Each pixel is a 1-bit quantity holding an alpha value. Pixels are packed together into 32-bit quantities. The ordering of the bits matches the endianness of the platform. On a big-endian machine, the first pixel is in the uppermost bit, on a little-endian machine the first pixel is in the least-significant bit.

`#CAIRO_FORMAT_RGB16_565`
: Each pixel is a 16-bit quantity with red in the upper 5 bits, then green in the middle 6 bits, and blue in the lower 5 bits.

`#CAIRO_FORMAT_RGB30`
: Like RGB24 but with 10bpc.

This function returns a handle to the newly created surface. The caller owns the surface and should call csurface:Free() when done with it.

This function always returns a valid handle, but it will return a handle to a "nil" surface if an error such as out of memory occurs. You can use csurface:Status() to check for this.

INPUTS

`format` format of pixels in the surface to create (see above)

`width` width of the surface, in pixels

`height` height of the surface, in pixels

RESULTS

`handle` image surface

4.6 cairo.Path

NAME

cairo.Path – construct new path

SYNOPSIS

```
p = cairo.Path(table)
```

FUNCTION

Constructs a new Cairo path from a table source. The table can contain a number of subtables describing a path item each. Each subtable inside `table` must contain a `Type` field that specifies the path item type of the subtable. Additional subtable items that must be initialized depend on the item type set in the `Type` field.

The following items are recognized for each subtable within `table`:

`Type` Describes the type of path item. This can be one of the following types:

`#CAIRO_PATH_MOVE_TO`
: A move-to operation. You must set the fields `x1` and `y1` as well.

`#CAIRO_PATH_LINE_TO`
: A line-to operation. You must set the fields `x1` and `y1` as well.

`#CAIRO_PATH_CURVE_TO`
: A curve-to operation. You must set the fields `x1`, `y1`, `x2`, `y2`, `x3` and `y3` as well.

`#CAIRO_PATH_CLOSE_PATH`
: A close-path operation. No additional fields need to be set.

`x1, y1` Control points needed by `#CAIRO_PATH_MOVE_TO`, `#CAIRO_PATH_LINE_TO`, and `#CAIRO_PATH_CURVE_TO`.

`x2, y2, x3, y3`
: Additional control points, only needed by `#CAIRO_PATH_CURVE_TO`.

INPUTS

`table` table describing the path

RESULTS

`p` a newly allocated Cairo path

4.7 cairo.ScaledFont

NAME

cairo.ScaledFont – create scaled font

SYNOPSIS

```
font = cairo.ScaledFont(font_face, font_matrix, ctm, options)
```

FUNCTION

Creates a Cairo scaled font object from a font face and matrices that describe the size of the font and the environment in which it will be used. The `options` argument must be a Cairo font options handle created with cairo.FontOptions() or cfontoptions:Copy() The `font_matrix` and `ctm` arguments must be Cairo matrices.

In the simplest case of a `N` point font, `font_matrix` is just a scale by `N`, but it can also be used to shear the font or stretch it unequally along the two axes.

This function returns a newly created Cairo scaled font. Destroy with cscaledfont:Free()

INPUTS

`font_face` a Cairo font face

`font_matrix` font space to user space transformation matrix for the font

`ctm` user to device transformation matrix with which the font will be used

`options` options to use when getting metrics for the font and rendering with it

RESULTS

`font` a newly created Cairo scaled font

4.8 cairo.ImageSurfaceFromBrush

NAME

cairo.ImageSurfaceFromBrush – create image surface from Hollywood brush

SYNOPSIS

```
handle = cairo.ImageSurfaceFromBrush(id)
```

FUNCTION

Creates an image surface from the Hollywood brush specified by `id`. If the brush has transparent areas, the image surface will use `#CAIRO_FORMAT_ARGB32`, otherwise it will use the `#CAIRO_FORMAT_RGB24` pixel format.

This function returns a handle to the newly created surface. The caller owns the surface and should call csurface:Free() when done with it.

INPUTS

`id`	Hollywood brush to convert to an image surface

RESULTS

`handle`	image surface

4.9 cairo.Matrix

NAME

cairo.Matrix – create matrix

SYNOPSIS

```
m = cairo.Matrix([xx, yx, xy, yy, x0, y0])
```

FUNCTION

Creates a matrix and optionally initializes its affine transformation to the coefficients specified by `xx`, `yx`, `xy`, `yy`, `x0`, `y0`. Omitted coefficients will be set to 0.

INPUTS

`xx`	optional: xx component of the affine transformation (defaults to 0)
`yx`	optional: yx component of the affine transformation (defaults to 0)
`xy`	optional: xy component of the affine transformation (defaults to 0)
`yy`	optional: yy component of the affine transformation (defaults to 0)
`x0`	optional: X translation component of the affine transformation (defaults to 0)
`y0`	optional: Y translation component of the affine transformation (defaults to 0)

RESULTS

`m`	matrix object

4.10 cairo.MatrixIdentity

NAME

cairo.MatrixIdentity – create identity matrix

SYNOPSIS

```
m = cairo.MatrixIdentity()
```

FUNCTION

Creates a matrix and initializes its affine transformation to an identity transformation.

INPUTS

none

RESULTS

`m`	identity matrix object

4.11 cairo.PatternForSurface

NAME

cairo.PatternForSurface – create pattern for surface

SYNOPSIS

```
pat = cairo.PatternForSurface(surface)
```

FUNCTION

Create a new Cairo pattern for the given surface.

This function returns the newly created Cairo pattern if successful, or an error pattern in case of no memory. The caller owns the returned object and should call cpattern:Free() when finished with it.

This function will always return a valid handle, but if an error occurred the pattern status will be set to an error. To inspect the status of a pattern use cpattern:Status().

INPUTS

`surface`	the surface

RESULTS

`pat`	the newly created Cairo pattern

4.12 cairo.PatternLinear

NAME

cairo.PatternLinear – create linear pattern

SYNOPSIS

```
pat = cairo.PatternLinear(x0, y0, x1, y1)
```

FUNCTION

Create a new linear gradient Cairo pattern along the line defined by (x0, y0) and (x1, y1). Before using the gradient pattern, a number of color stops should be defined using cpattern:AddColorStopRGB() or cpattern:AddColorStopRGBA().

Note: The coordinates here are in pattern space. For a new pattern, pattern space is identical to user space, but the relationship between the spaces can be changed with cpattern:SetMatrix().

This function returns the newly created Cairo pattern if successful, or an error pattern in case of no memory. The caller owns the returned object and should call cpattern:Free() when finished with it.

This function will always return a valid handle, but if an error occurred the pattern status will be set to an error. To inspect the status of a pattern use cpattern:Status().

INPUTS

`x0`	x coordinate of the start point
`y0`	y coordinate of the start point
`x1`	x coordinate of the end point

`y1` y coordinate of the end point

RESULTS

`pat` the newly created Cairo pattern

4.13 cairo.PatternMesh

NAME

cairo.PatternMesh – create mesh pattern

SYNOPSIS

```
pat = cairo.PatternMesh()
```

FUNCTION

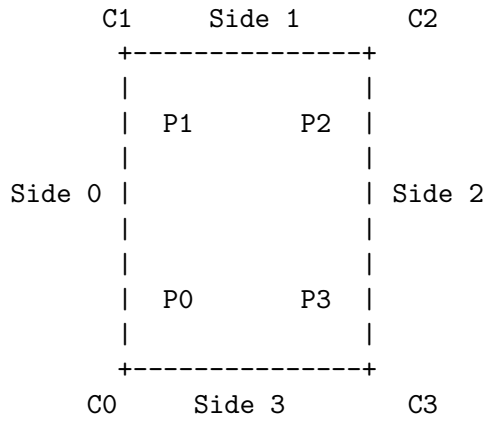
Create a new mesh pattern.

Mesh patterns are tensor-product patch meshes (type 7 shadings in PDF). Mesh patterns may also be used to create other types of shadings that are special cases of tensor-product patch meshes such as Coons patch meshes (type 6 shading in PDF) and Gouraud-shaded triangle meshes (type 4 and 5 shadings in PDF).

Mesh patterns consist of one or more tensor-product patches, which should be defined before using the mesh pattern. Using a mesh pattern with a partially defined patch as source or mask will put the context in an error status with a status of `#CAIRO_STATUS_INVALID_MESH_CONSTRUCTION`.

A tensor-product patch is defined by 4 Bezier curves (side 0, 1, 2, 3) and by 4 additional control points (P0, P1, P2, P3) that provide further control over the patch and complete the definition of the tensor-product patch. The corner C0 is the first point of the patch.

Degenerate sides are permitted so straight lines may be used. A zero length line on one side may be used to create 3 sided patches.



Each patch is constructed by first calling cpattern:BeginPatch(), then cpattern:MoveTo() to specify the first point in the patch (C0). Then the sides are specified with calls to cpattern:CurveTo() and cpattern:LineTo().

The four additional control points (P0, P1, P2, P3) in a patch can be specified with cpattern:SetControlPoint().

At each corner of the patch (C0, C1, C2, C3) a color may be specified with cpattern:SetCornerColorRGB() or cpattern:SetCornerColorRGBA(). Any corner whose color is not explicitly specified defaults to transparent black.

A Coons patch is a special case of the tensor-product patch where the control points are implicitly defined by the sides of the patch. The default value for any control point not specified is the implicit value for a Coons patch, i.e. if no control points are specified the patch is a Coons patch.

A triangle is a special case of the tensor-product patch where the control points are implicitly defined by the sides of the patch, all the sides are lines and one of them has length 0, i.e. if the patch is specified using just 3 lines, it is a triangle. If the corners connected by the 0-length side have the same color, the patch is a Gouraud-shaded triangle.

Patches may be oriented differently to the above diagram. For example the first point could be at the top left. The diagram only shows the relationship between the sides, corners and control points. Regardless of where the first point is located, when specifying colors, corner 0 will always be the first point, corner 1 the point between side 0 and side 1 etc.

Calling cpattern:EndPatch() completes the current patch. If less than 4 sides have been defined, the first missing side is defined as a line from the current point to the first point of the patch (C0) and the other sides are degenerate lines from C0 to C0. The corners between the added sides will all be coincident with C0 of the patch and their color will be set to be the same as the color of C0.

Additional patches may be added with additional calls to cpattern:BeginPatch() / cpattern:EndPatch().

```
pat = cairo.PatternMesh()

; Add a Coons patch
pat:BeginPatch()
pat:MoveTo(0, 0)
pat:CurveTo(30, -30, 60, 30, 100, 0)
pat:CurveTo(60, 30, 130, 60, 100, 100)
pat:CurveTo(60, 70, 30, 130,  0, 100)
pat:CurveTo(30, 70, -30, 30,  0, 0)
pat:SetCornerColorRGB(0, 1, 0, 0)
pat:SetCornerColorRGB(1, 0, 1, 0)
pat:SetCornerColorRGB(2, 0, 0, 1)
pat:SetCornerColorRGB(3, 1, 1, 0)
pat:EndPatch()

 Add a Gouraud-shaded triangle
pat:BeginPatch()
pat:MoveTo(100, 100)
pat:LineTo(130, 130)
pat:LineTo(130, 70)
pat:SetCornerColorRGB(0, 1, 0, 0)
pat:SetCornerColorRGB(1, 0, 1, 0)
```

```
pat:SetCornerColorRGB(2, 0, 0, 1)
pat:EndPatch()
```

When two patches overlap, the last one that has been added is drawn over the first one.

When a patch folds over itself, points are sorted depending on their parameter coordinates inside the patch. The v coordinate ranges from 0 to 1 when moving from side 3 to side 1; the u coordinate ranges from 0 to 1 when going from side 0 to side 2. Points with higher v coordinate hide points with lower v coordinate. When two points have the same v coordinate, the one with higher u coordinate is above. This means that points nearer to side 1 are above points nearer to side 3; when this is not sufficient to decide which point is above (for example when both points belong to side 1 or side 3) points nearer to side 2 are above points nearer to side 0.

For a complete definition of tensor-product patches, see the PDF specification (ISO32000), which describes the parametrization in detail.

Note: The coordinates are always in pattern space. For a new pattern, pattern space is identical to user space, but the relationship between the spaces can be changed with cpattern:SetMatrix().

This function returns the newly created Cairo pattern if successful, or an error pattern in case of no memory. The caller owns the returned object and should call cpattern:Free() when finished with it.

This function will always return a valid handle, but if an error occurred the pattern status will be set to an error. To inspect the status of a pattern use cpattern:Status().

INPUTS

none

RESULTS

`pat` the newly created Cairo pattern

4.14 cairo.PatternRadial

NAME

cairo.PatternRadial – create radial pattern

SYNOPSIS

```
pat = cairo.PatternRadial(cx0, cy0, radius0, cx1, cy1, radius1)
```

FUNCTION

Creates a new radial gradient Cairo pattern between the two circles defined by (cx0, cy0, radius0) and (cx1, cy1, radius1). Before using the gradient pattern, a number of color stops should be defined using cpattern:AddColorStopRGB() or cpattern:AddColorStopRGBA().

Note: The coordinates here are in pattern space. For a new pattern, pattern space is identical to user space, but the relationship between the spaces can be changed with cpattern:SetMatrix().

This function returns the newly created Cairo pattern if successful, or an error pattern in case of no memory. The caller owns the returned object and should call cpattern:Free() when finished with it.

This function will always return a valid handle, but if an error occurred the pattern status will be set to an error. To inspect the status of a pattern use cpattern:Status().

INPUTS

`cx0`	x coordinate for the center of the start circle
`cy0`	y coordinate for the center of the start circle
`radius0`	radius of the start circle
`cx1`	x coordinate for the center of the end circle
`cy1`	y coordinate for the center of the end circle
`radius1`	radius of the end circle

RESULTS

`x`	the newly created Cairo pattern

4.15 cairo.PatternRGB

NAME

cairo.PatternRGB – create RGB pattern

SYNOPSIS

```
pat = cairo.PatternRGB(red, green, blue)
```

FUNCTION

Creates a new Cairo pattern corresponding to an opaque color. The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

The color is specified in the same way as in ccontext:SetSourceRGB().

This function returns the newly created Cairo pattern if successful, or an error pattern in case of no memory. The caller owns the returned object and should call cpattern:Free() when finished with it.

This function will always return a valid handle, but if an error occurred the pattern status will be set to an error. To inspect the status of a pattern use cpattern:Status().

INPUTS

`red`	red component of the color
`green`	green component of the color
`blue`	blue component of the color

RESULTS

`pat`	the newly created Cairo pattern

4.16 cairo.PatternRGBA

NAME

cairo.PatternRGBA – create RGBA pattern

SYNOPSIS

```
pat = cairo.PatternRGBA(red, green, blue, alpha)
```

FUNCTION

Creates a new Cairo pattern corresponding to a translucent color. The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

The color is specified in the same way as in ccontext:SetSourceRGB().

This function returns the newly created Cairo pattern if successful, or an error pattern in case of no memory. The caller owns the returned object and should call cpattern:Free() when finished with it.

This function will always return a valid handle, but if an error occurred the pattern status will be set to an error. To inspect the status of a pattern use cpattern:Status().

INPUTS

`red`	red component of the color
`green`	green component of the color
`blue`	blue component of the color
`alpha`	alpha component of the color

RESULTS

`pat`	the newly created Cairo pattern

4.17 cairo.PDFSurface

NAME

cairo.PDFSurface – create PDF surface

SYNOPSIS

```
handle = cairo.PDFSurface(filename$, width_in_points, height_in_points)
```

FUNCTION

Creates a PDF surface of the specified size in points to be written to `filename$`.

This function returns a handle to the newly created surface. The caller owns the surface and should call csurface:Free() when done with it.

This function always returns a valid handle, but it will return a handle to a "nil" surface if an error such as out of memory occurs. You can use csurface:Status() to check for this.

INPUTS

`filename$`
: a filename for the PDF output (must be writable)

`width_in_points`
: width of the surface, in points (1 point == 1/72.0 inch)

`height_in_points`
: height of the surface, in points (1 point == 1/72.0 inch)

RESULTS

`handle` PDF surface

4.18 cairo.PDFVersionToString

NAME

cairo.PDFVersionToString – get string from PDF version

SYNOPSIS

```
s$ = cairo.PDFVersionToString(version)
```

FUNCTION

Get the string representation of the given `version` id. This function will return `Nil` if `version` isn't valid. `version` can be one of the following constants:

```
#CAIRO_PDF_VERSION_1_4
#CAIRO_PDF_VERSION_1_5
```

This function returns the string associated to the given version.

INPUTS

`version` a version id

RESULTS

`s$` string representation of given version

4.19 cairo.Region

NAME

cairo.Region – create region

SYNOPSIS

```
reg = cairo.Region([rects])
```

FUNCTION

Allocates a new region object and optionally initializes it to the union of all given `rects`. If specified, the `rects` parameter must be a table containing an arbitrary number of subtables describing a rectangle each. The subtables need to have the fields `x`, `y`, `width`, `height` initialized. If the `rects` parameter is omitted, an empty region will be created.

This function returns a newly allocated Cairo region. Free with cregion:Free(). This function always returns a valid handle; if memory cannot be allocated, then a special error object is returned where all operations on the object do nothing. You can check for this with cregion:Status().

INPUTS

`rects` optional: table of rectangles to initialize the region with

RESULTS

`reg` a newly allocated Cairo region

4.20 cairo.StatusToString

NAME

cairo.StatusToString – get string from status code

SYNOPSIS

```
s$ = cairo.StatusToString(status)
```

FUNCTION

Provides a human-readable description of the status code specified by `status`. See ccontext:Status() for a list of status codes.

INPUTS

`status` a Cairo status code

RESULTS

`s$` string representation of status code

4.21 cairo.SVGSurface

NAME

cairo.SVGSurface – create SVG surface

SYNOPSIS

```
handle = cairo.SVGSurface(filename$, width_in_points, height_in_points)
```

FUNCTION

Creates a SVG surface of the specified size in points to be written to `filename$`.

The SVG surface backend recognizes the following MIME types for the data attached to a surface (see csurface:SetMimeData()) when it is used as a source pattern for drawing on this surface: `#CAIRO_MIME_TYPE_JPEG`, `#CAIRO_MIME_TYPE_PNG`, `#CAIRO_MIME_TYPE_URI`. If any of them is specified, the SVG backend emits a href with the content of MIME data instead of a surface snapshot (PNG, Base64-encoded) in the corresponding image tag.

The unofficial MIME type `#CAIRO_MIME_TYPE_URI` is examined first. If present, the URI is emitted as is: assuring the correctness of URI is left to the client code.

If `#CAIRO_MIME_TYPE_URI` is not present, but `#CAIRO_MIME_TYPE_JPEG` or `#CAIRO_MIME_TYPE_PNG` is specified, the corresponding data is Base64-encoded and emitted.

If `#CAIRO_MIME_TYPE_UNIQUE_ID` is present, all surfaces with the same unique identifier will only be embedded once.

This function returns a handle to the newly created surface. The caller owns the surface and should call csurface:Free() when done with it.

This function always returns a valid handle, but it will return a handle to a "nil" surface if an error such as out of memory occurs. You can use csurface:Status() to check for this.

INPUTS

`filename$`
a filename for the SVG output (must be writable)

`width_in_points`
width of the surface, in points (1 point == 1/72.0 inch)

`height_in_points`
height of the surface, in points (1 point == 1/72.0 inch)

RESULTS

`handle` SVG surface

4.22 cairo.SVGVersionToString

NAME

cairo.SVGVersionToString – get string from SVG version

SYNOPSIS

```
s$ = cairo.SVGVersionToString(version)
```

FUNCTION

Get the string representation of the given `version` id. This function will return `Nil` if `version` isn't valid. `version` can be one of the following constants:

```
#CAIRO_SVG_VERSION_1_1
#CAIRO_SVG_VERSION_1_2
```

This function returns the string associated to given version.

INPUTS

`version` a version id

RESULTS

`s$` string representation of given version

4.23 cairo.ToyFontFace

NAME

cairo.ToyFontFace – create toy font face

SYNOPSIS

```
font = cairo.ToyFontFace(family$, slant, weight)
```

FUNCTION

Creates a font face from a triplet of `family$`, `slant`, and `weight`. These font faces are used in implementation of the Cairo context "toy" font API.

The `slant` parameter can be one of the following constants:

```
#CAIRO_FONT_SLANT_NORMAL
#CAIRO_FONT_SLANT_ITALIC
#CAIRO_FONT_SLANT_OBLIQUE
```

The `weight` parameter can be one of the following constants:

```
#CAIRO_FONT_WEIGHT_NORMAL
#CAIRO_FONT_WEIGHT_BOLD
```

If `family$` is the zero-length string "", the platform-specific default family is assumed. The default family then can be queried using cfontface:GetFamily().

The ccontext:SelectFontFace() function uses this to create font faces. See that function for limitations and other details of toy font faces.

This function returns a newly created Cairo font face. Free with cfontface:Free() when you are done using it.

INPUTS

`family$`	a font family name
`slant`	the slant for the font (see above)
`weight`	the weight for the font (see above)

RESULTS

`font`	a newly created Cairo font face

4.24 cairo.Version

NAME

cairo.Version – get Cairo version

SYNOPSIS

```
ver$ = cairo.MatrixIdentity()
```

FUNCTION

Returns the version of the cairo library as a human-readable string of the form "X.Y.Z".

INPUTS

none

RESULTS

`ver$`	Cairo version used by Pangomonium

5 Cairo context

5.1 ccontext:AppendPath

NAME

ccontext:AppendPath – append path

SYNOPSIS

```
ccontext:AppendPath(path)
```

FUNCTION

Append the `path` onto the current path. The `path` may be either the return value from one of ccontext:CopyPath() or ccontext:CopyPathFlat() or it may be constructed using cairo.Path().

INPUTS

`path`	path to be appended

5.2 ccontext:Arc

NAME

ccontext:Arc – add arc to current path

SYNOPSIS

```
ccontext:Arc(xc, yc, radius, angle1, angle2)
```

FUNCTION

Adds a circular arc of the given `radius` to the current path. The arc is centered at (`xc`, `yc`), begins at `angle1` and proceeds in the direction of increasing angles to end at `angle2`. If `angle2` is less than `angle1` it will be progressively increased by 2*#PI until it is greater than `angle1`.

If there is a current point, an initial line segment will be added to the path to connect the current point to the beginning of the arc. If this initial line is undesired, it can be avoided by calling ccontext:NewSubPath() before calling ccontext:Arc().

Angles are measured in radians. An angle of 0.0 is in the direction of the positive X axis (in user space). An angle of #PI/2.0 radians (90 degrees) is in the direction of the positive Y axis (in user space). Angles increase in the direction from the positive X axis toward the positive Y axis. So with the default transformation matrix, angles increase in a clockwise direction.

To convert from degrees to radians, use degrees * `#PI` / 180.

This function gives the arc in the direction of increasing angles; see ccontext:ArcNegative() to get the arc in the direction of decreasing angles.

The arc is circular in user space. To achieve an elliptical arc, you can scale the current transformation matrix by different amounts in the X and Y directions. For example, to draw an ellipse in the box given by `x`, `y`, `width`, `height`:

```
ctx:Save()
```

```
ctx:Translate(x + width / 2, y + height / 2)
ctx:Scale(width / 2, height / 2)
ctx:Arc(0, 0, 1, 0, 2 * #PI)
ctx:Restore()
```

INPUTS

`xc`	X position of the center of the arc
`yc`	Y position of the center of the arc
`radius`	the radius of the arc
`angle1`	the start angle, in radians
`angle2`	the end angle, in radians

5.3 ccontext:ArcNegative

NAME

ccontext:ArcNegative – add arc to current path

SYNOPSIS

`ccontext:ArcNegative(xc, yc, radius, angle1, angle2)`

FUNCTION

Adds a circular arc of the given `radius` to the current path. The arc is centered at (`xc`, `yc`), begins at `angle1` and proceeds in the direction of decreasing angles to end at `angle2`. If `angle2` is greater than `angle1` it will be progressively decreased by 2*#PI until it is less than `angle1`.

See ccontext:Arc() for more details. This function differs only in the direction of the arc between the two angles.

INPUTS

`xc`	X position of the center of the arc
`yc`	Y position of the center of the arc
`radius`	the radius of the arc
`angle1`	the start angle, in radians
`angle2`	the end angle, in radians

5.4 ccontext:Clip

NAME

ccontext:Clip – set new clip region

SYNOPSIS

`ccontext:Clip()`

FUNCTION

Establishes a new clip region by intersecting the current clip region with the current path as it would be filled by ccontext:Fill() and according to the current fill rule (see ccontext:SetFillRule()).

After ccontext:Clip(), the current path will be cleared from the Cairo context.

The current clip region affects all drawing operations by effectively masking out any changes to the surface that are outside the current clip region.

Calling ccontext:Clip() can only make the clip region smaller, never larger. But the current clip is part of the graphics state, so a temporary restriction of the clip region can be achieved by calling ccontext:Clip() within a ccontext:Save() / ccontext:Restore() pair. The only other means of increasing the size of the clip region is ccontext:ResetClip().

INPUTS

none

5.5 ccontext:ClipExtents

NAME

ccontext:ClipExtents – get clip extents

SYNOPSIS

```
x1, y1, x2, y2 = ccontext:ClipExtents()
```

FUNCTION

Computes a bounding box in user coordinates covering the area inside the current clip.

INPUTS

none

RESULTS

`x1`	left of the resulting extents
`y1`	top of the resulting extents
`x2`	right of the resulting extents
`y2`	bottom of the resulting extents

5.6 ccontext:ClipPreserve

NAME

ccontext:ClipPreserve – set new clip region and preserve path

SYNOPSIS

```
ccontext:ClipPreserve()
```

FUNCTION

Establishes a new clip region by intersecting the current clip region with the current path as it would be filled by ccontext:Fill() and according to the current fill rule (see ccontext:SetFillRule()).

Unlike ccontext:Clip(), ccontext:ClipPreserve() preserves the path within the Cairo context.

The current clip region affects all drawing operations by effectively masking out any changes to the surface that are outside the current clip region.

Calling ccontext:ClipPreserve() can only make the clip region smaller, never larger. But the current clip is part of the graphics state, so a temporary restriction of the clip region can be achieved by calling ccontext:ClipPreserve() within a ccontext:Save() / ccontext:Restore() pair. The only other means of increasing the size of the clip region is ccontext:ResetClip().

INPUTS

none

5.7 ccontext:ClosePath

NAME

ccontext:ClosePath – close path

SYNOPSIS

```
ccontext:ClosePath()
```

FUNCTION

Adds a line segment to the path from the current point to the beginning of the current sub-path, (the most recent point passed to ccontext:MoveTo()), and closes this sub-path. After this call the current point will be at the joined endpoint of the sub-path.

The behavior of ccontext:ClosePath() is distinct from simply calling ccontext:LineTo() with the equivalent coordinate in the case of stroking. When a closed sub-path is stroked, there are no caps on the ends of the sub-path. Instead, there is a line join connecting the final and initial segments of the sub-path.

If there is no current point before the call to ccontext:ClosePath(), this function will have no effect.

Note: Any call to ccontext:ClosePath() will place an explicit `MOVE_TO` element into the path immediately after the `CLOSE_PATH` element, which can be seen in ccontext:CopyPath() for example. This can simplify path processing in some cases as it may not be necessary to save the "last move_to point" during processing as the `MOVE_TO` immediately after the `CLOSE_PATH` will provide that point.

INPUTS

none

5.8 ccontext:CopyPage

NAME

ccontext:CopyPage – copy page

SYNOPSIS

```
ccontext:CopyPage()
```

FUNCTION

Emits the current page for backends that support multiple pages, but doesn't clear it, so, the contents of the current page will be retained for the next page too. Use ccontext:ShowPage() if you want to get an empty page after the emission.

This is a convenience function that simply calls csurface:CopyPage() on the Cairo context's target.

INPUTS

none

5.9 ccontext:CopyPath

NAME

ccontext:CopyPath – copy path

SYNOPSIS

```
handle = ccontext:CopyPath()
```

FUNCTION

Creates a copy of the current path and returns it to the user as a Cairo path object. Use cpath:Get() to get the individual path members.

This function will always return a valid handle, but the result will have no data, if either of the following conditions hold:

1. If there is insufficient memory to copy the path. In this case the path status will be set to `#CAIRO_STATUS_NO_MEMORY`.
2. If the Cairo context is already in an error state. In this case the path status will contain the same status that would be returned by ccontext:Status().

This function returns the copy of the current path. The caller owns the returned object and should call cpath:Free() when finished with it.

INPUTS

none

RESULTS

`handle` the copy of the current path

5.10 ccontext:CopyPathFlat

NAME

ccontext:CopyPathFlat – copy flattened version of path

SYNOPSIS

```
handle = ccontext:CopyPathFlat()
```

FUNCTION

Gets a flattened copy of the current path and returns it to the user as a Cairo path object. Use cpath:Get() to get the individual path members.

This function is like ccontext:CopyPath() except that any curves in the path will be approximated with piecewise-linear approximations, (accurate to within the current tolerance value). That is, the result is guaranteed to not have any elements of type `#CAIRO_PATH_CURVE_TO` which will instead be replaced by a series of `#CAIRO_PATH_LINE_TO` elements.

This function will always return a valid handle, but the result will have no data, if either of the following conditions hold:

1. If there is insufficient memory to copy the path. In this case the path status will be set to `#CAIRO_STATUS_NO_MEMORY`.
2. If the Cairo context is already in an error state. In this case the path status will contain the same status that would be returned by ccontext:Status().

This function returns the copy of the current path. The caller owns the returned object and should call cpath:Free() when finished with it.

INPUTS

none

RESULTS

`handle` the copy of the current path

5.11 ccontext:CurveTo

NAME

ccontext:CurveTo – add cubic Bezier spline to path

SYNOPSIS

```
ccontext:CurveTo(x1, y1, x2, y2, x3, y3)
```

FUNCTION

Adds a cubic Bezier spline to the path from the current point to position (`x3`, `y3`) in user-space coordinates, using (`x1`, `y1`) and (`x2`, `y2`) as the control points. After this call the current point will be (`x3`, `y3`).

If there is no current point before the call to ccontext:CurveTo() this function will behave as if preceded by a call to ccontext:MoveTo() with `x1` and `y1`.

INPUTS

`x1` the X coordinate of the first control point

`y1` the Y coordinate of the first control point

`x2` the X coordinate of the second control point

`y2` the Y coordinate of the second control point

`x3` the X coordinate of the end of the curve

`y3` the Y coordinate of the end of the curve

5.12 ccontext:DeviceToUser

NAME

ccontext:DeviceToUser – device to user

SYNOPSIS

```
ux, uy = ccontext:DeviceToUser(dx, dy)
```

FUNCTION

Transform a coordinate from device space to user space by multiplying the given point by the inverse of the current transformation matrix (CTM).

INPUTS

`dx`	X value of coordinate (device space)
`dy`	Y value of coordinate (device space)

RESULTS

`ux`	X value of coordinate (user space)
`uy`	Y value of coordinate (user space)

5.13 ccontext:DeviceToUserDistance

NAME

ccontext:DeviceToUserDistance – device to user distance

SYNOPSIS

```
ux, uy = ccontext:DeviceToUserDistance(dx, dy)
```

FUNCTION

Transform a distance vector from device space to user space. This function is similar to ccontext:DeviceToUser() except that the translation components of the inverse CTM will be ignored when transforming (`dx`, `dy`).

INPUTS

`dx`	X component of a distance vector (device space)
`dy`	Y component of a distance vector (device space)

INPUTS

`ux`	X component of a distance vector (user space)
`uy`	Y component of a distance vector (user space)

5.14 ccontext:ErrorUnderlinePath

NAME

ccontext:ErrorUnderlinePath – add squiggly line to path

SYNOPSIS

```
ccontext:ErrorUnderlinePath(x, y, width, height)
```

FUNCTION

Add a squiggly line to the current path in the specified Cairo context that approximately covers the given rectangle in the style of an underline used to indicate a spelling error.

The width of the underline is rounded to an integer number of up/down segments and the resulting rectangle is centered in the original rectangle.

INPUTS

`x`	the X coordinate of one corner of the rectangle
`y`	the Y coordinate of one corner of the rectangle
`width`	non-negative width of the rectangle
`height`	non-negative height of the rectangle

5.15 ccontext:Fill

NAME

ccontext:Fill – fill current path

SYNOPSIS

```
ccontext:Fill()
```

FUNCTION

A drawing operator that fills the current path according to the current fill rule. Each subpath is implicitly closed before being filled. After ccontext:Fill(), the current path will be cleared from the Cairo context. See ccontext:SetFillRule() and ccontext:FillPreserve().

INPUTS

none

5.16 ccontext:FillExtents

NAME

ccontext:FillExtents – get fill extents

SYNOPSIS

```
x1, y1, x2, y2 = ccontext:FillExtents()
```

FUNCTION

Computes a bounding box in user coordinates covering the area that would be affected (the "inked" area) by a ccontext:Fill() operation given the current path and fill parameters. If the current path is empty, returns an empty rectangle ((0,0), (0,0)). Surface dimensions and clipping are not taken into account.

Contrast with ccontext:PathExtents(), which is similar, but returns non-zero extents for some paths with no inked area, such as a simple line segment.

Note that ccontext:FillExtents() must necessarily do more work to compute the precise inked areas in light of the fill rule, so ccontext:PathExtents() may be more desirable for sake of performance if the non-inked path extents are desired.

See ccontext:Fill(), ccontext:SetFillRule() and ccontext:FillPreserve().

INPUTS
none

RESULTS

`x1`	left of the resulting extents
`y1`	top of the resulting extents
`x2`	right of the resulting extents
`y2`	bottom of the resulting extents

5.17 ccontext:FillPreserve

NAME
ccontext:FillPreserve – fill and preserve path

SYNOPSIS
```
ccontext:FillPreserve()
```

FUNCTION
A drawing operator that fills the current path according to the current fill rule. Each subpath is implicitly closed before being filled. Unlike ccontext:Fill(), ccontext:FillPreserve() preserves the path within the Cairo context.

See ccontext:SetFillRule() and ccontext:Fill().

INPUTS
none

5.18 ccontext:FontExtents

NAME
ccontext:FontExtents – font extents

SYNOPSIS
```
t = ccontext:FontExtents()
```

FUNCTION
Gets the font extents for the currently selected font. This will return a table that has the following elements initialized:

`Ascent`	The distance that the font extends above the baseline. Note that this is not always exactly equal to the maximum of the extents of all the glyphs in the font, but rather is picked to express the font designer's intent as to how the font should align with elements above it.
`Descent`	The distance that the font extends below the baseline. This value is positive for typical fonts that include portions below the baseline. Note that this is not always exactly equal to the maximum of the extents of all the glyphs in the font, but rather is picked to express the font designer's intent as to how the font should align with elements below it.

`Height` The recommended vertical distance between baselines when setting consecutive lines of text with the font. This is greater than `Ascent + Descent` by a quantity known as the line spacing or external leading. When space is at a premium, most fonts can be set with only a distance of `Ascent + Descent` between lines.

`MaxXAdvance`
The maximum distance in the X direction that the origin is advanced for any glyph in the font.

`MaxYAdvance`
The maximum distance in the Y direction that the origin is advanced for any glyph in the font. This will be zero for normal fonts used for horizontal writing (The scripts of East Asia are sometimes written vertically.)

All values are given in the current user-space coordinate system.

Because font metrics are in user-space coordinates, they are mostly, but not entirely, independent of the current transformation matrix. If you call ccontext:Scale() with coefficients of (2.0, 2.0), text will be drawn twice as big, but the reported text extents will not be doubled. They will change slightly due to hinting (so you can't assume that metrics are independent of the transformation matrix), but otherwise will remain unchanged.

INPUTS
none

RESULTS
`t` table containing the font extents (see above)

5.19 ccontext:Free

NAME
ccontext:Free – destroy context

SYNOPSIS
`ccontext:Free()`

FUNCTION
Decreases the reference count on the Cairo context by one. If the result is zero, then the Cairo context and all associated resources are freed. See ccontext:Reference().

INPUTS
none

5.20 ccontext:GetAntialias

NAME
ccontext:GetAntialias – get current antialias mode

SYNOPSIS

```
mode = ccontext:GetAntialias()
```

FUNCTION

Gets the current shape antialiasing mode, as set by ccontext:SetAntialias().

See ccontext:SetAntialias() for a list of antialiasing modes.

INPUTS

none

RESULTS

`mode` the current shape antialiasing mode

5.21 ccontext:GetCurrentPoint

NAME

ccontext:GetCurrentPoint – get current point

SYNOPSIS

```
x, y = ccontext:GetCurrentPoint()
```

FUNCTION

Gets the current point of the current path, which is conceptually the final point reached by the path so far.

The current point is returned in the user-space coordinate system. If there is no defined current point or if the context is in an error status, `x` and `y` will both be set to 0.0. It is possible to check this in advance with ccontext:HasCurrentPoint().

Most path construction functions alter the current point. See the following for details on how they affect the current point: ccontext:NewPath(), ccontext:NewSubPath(), ccontext:AppendPath(), ccontext:ClosePath(), ccontext:MoveTo(), ccontext:LineTo(), ccontext:CurveTo(), ccontext:RelMoveTo(), ccontext:RelLineTo(), ccontext:RelCurveTo(), ccontext:Arc(), ccontext:ArcNegative(), ccontext:Rectangle(), ccontext:TextPath().

Some functions use and alter the current point but do not otherwise change current path: ccontext:ShowText().

Some functions unset the current path and as a result, current point: ccontext:Fill(), ccontext:Stroke().

INPUTS

none

RESULTS

`x` return value for X coordinate of the current point

`y` return value for Y coordinate of the current point

5.22 ccontext:GetDash

NAME
ccontext:GetDash – get current dash array

SYNOPSIS
```
dashes, offset = ccontext:GetDash()
```

FUNCTION
Gets the current dash array.

INPUTS
none

RESULTS

`dashes`	current dash array
`offset`	current dash offset

5.23 ccontext:GetDashCount

NAME
ccontext:GetDashCount – get current dash count

SYNOPSIS
```
count = ccontext:GetDashCount()
```

FUNCTION
This function returns the length of the dash array in the Cairo context or 0 if dashing is not currently in effect.

See also ccontext:SetDash() and ccontext:GetDash().

INPUTS
none

RESULTS

`count`	the length of the dash array, or 0 if no dash array set

5.24 ccontext:GetFillRule

NAME
ccontext:GetFillRule – get current fill rule

SYNOPSIS
```
fillrule = ccontext:GetFillRule()
```

FUNCTION
Gets the current fill rule, as set by ccontext:SetFillRule(). See ccontext:SetFillRule() for a list of fill rules.

INPUTS
none

RESULTS

`fillrule` the current fill rule

5.25 ccontext:GetFontFace

NAME

ccontext:GetFontFace – get current font face

SYNOPSIS

```
face = ccontext:GetFontFace()
```

FUNCTION

Gets the current font face for a Cairo context.

This function returns the current font face. This object is owned by Cairo. To keep a reference to it, you must call cfontface:Reference().

This function never returns `Nil`. If memory cannot be allocated, a special `Nil` font face object will be returned on which cfontface:Status() returns `#CAIRO_STATUS_NO_MEMORY`. Using this `Nil` object will cause its error state to propagate to other objects it is passed to, for example, calling ccontext:SetFontFace() with a `Nil` font will trigger an error that will shutdown the Cairo context object.

INPUTS

none

RESULTS

`face` the current font face

5.26 ccontext:GetFontMatrix

NAME

ccontext:GetFontMatrix – get current font matrix

SYNOPSIS

```
m = ccontext:GetFontMatrix(matrix)
```

FUNCTION

Returns the current font matrix as a Cairo matrix object. See ccontext:SetFontMatrix().

INPUTS

none

RESULTS

`m` matrix object

5.27 ccontext:GetFontOptions

NAME

ccontext:GetFontOptions – get font options

SYNOPSIS

```
ccontext:GetFontOptions(options)
```

FUNCTION

Retrieves font rendering options set via ccontext:SetFontOptions(). Note that the returned options do not include any options derived from the underlying surface; they are literally the options passed to ccontext:SetFontOptions().

INPUTS

`options` a Cairo font options object into which to store the retrieved options; all existing values are overwritten

5.28 ccontext:GetGroupTarget

NAME

ccontext:GetGroupTarget – get group target

SYNOPSIS

```
handle = ccontext:GetGroupTarget()
```

FUNCTION

Gets the current destination surface for the context. This is either the original target surface as passed to cairo.Context() or the target surface for the current group as started by the most recent call to ccontext:PushGroup() or ccontext:PushGroupWithContent().

This function will always return a valid handle, but the result can be a "nil" surface if the Cairo context is already in an error state. A `Nil` surface is indicated by csurface:Status() being different from `#CAIRO_STATUS_SUCCESS`.

This function returns the target surface. This object is owned by Cairo. To keep a reference to it, you must call csurface:Reference().

INPUTS

none

RESULTS

`handle` the target surface (owned by Cairo)

5.29 ccontext:GetLineCap

NAME

ccontext:GetLineCap – get current line cap style

SYNOPSIS

```
cap = ccontext:GetLineCap()
```

FUNCTION

Gets the current line cap style, as set by ccontext:SetLineCap(). See ccontext:SetLineCap() for a list of line cap styles.

INPUTS

none

RESULTS

`cap` the current line cap style

5.30 ccontext:GetLineJoin

NAME

ccontext:GetLineJoin – get current line join style

SYNOPSIS

```
join = ccontext:GetLineJoin()
```

FUNCTION

Gets the current line join style, as set by ccontext:SetLineJoin(). See ccontext:SetLineJoin() for a list of line join styles.

INPUTS

none

RESULTS

`join` the current line join style

5.31 ccontext:GetLineWidth

NAME

ccontext:GetLineWidth – get current line width

SYNOPSIS

```
width = ccontext:GetLineWidth()
```

FUNCTION

This function returns the current line width value exactly as set by ccontext:SetLineWidth(). Note that the value is unchanged even if the CTM has changed between the calls to ccontext:SetLineWidth() and ccontext:GetLineWidth().

INPUTS

none

RESULTS

`width` the current line width

5.32 ccontext:GetMatrix

NAME
ccontext:GetMatrix – get current transformation matrix

SYNOPSIS
```
m = ccontext:GetMatrix(matrix)
```

FUNCTION
Returns the current transformation matrix (CTM) as a Cairo matrix object. Use cmatrix:Get() to query the individual matrix coefficients.

INPUTS
none

RESULTS

`m`	current transformation matrix

5.33 ccontext:GetMiterLimit

NAME
ccontext:GetMiterLimit – get current miter limit

SYNOPSIS
```
limit = ccontext:GetMiterLimit()
```

FUNCTION
Gets the current miter limit, as set by ccontext:SetMiterLimit().

INPUTS
none

RESULTS

`limit`	the current miter limit

5.34 ccontext:GetOperator

NAME
ccontext:GetOperator – get current compositing operator

SYNOPSIS
```
op = ccontext:GetOperator()
```

FUNCTION
Gets the current compositing operator for a Cairo context. See ccontext:SetOperator() for a list of compositing operators.

INPUTS
none

RESULTS

`op`	the current compositing operator

5.35 ccontext:GetReferenceCount

NAME
ccontext:GetReferenceCount – get reference count

SYNOPSIS
```
count = ccontext:GetReferenceCount()
```

FUNCTION
Returns the current reference count of the context.

INPUTS
none

RESULTS

`count`	the current reference count

5.36 ccontext:GetScaledFont

NAME
ccontext:GetScaledFont – get current scaled font

SYNOPSIS
```
font = ccontext:GetScaledFont()
```

FUNCTION
Gets the current scaled font for a Cairo context.

This function returns the current scaled font. This object is owned by Cairo. To keep a reference to it, you must call cscaledfont:Reference().

This function never returns `Nil`. If memory cannot be allocated, a special "nil" scaled font object will be returned on which cscaledfont:Status() returns `#CAIRO_STATUS_NO_MEMORY`. Using this `Nil` object will cause its error state to propagate to other objects it is passed to, for example, calling ccontext:SetScaledFont() with a `Nil` font will trigger an error that will shutdown the Cairo context object.

INPUTS
none

RESULTS

`font`	the current scaled font (owned by Cairo)

5.37 ccontext:GetSource

NAME
ccontext:GetSource – get current source pattern

SYNOPSIS
```
pat = ccontext:GetSource()
```

FUNCTION

Gets the current source pattern for the Cairo context.

This function returns the current source pattern. This object is owned by Cairo. To keep a reference to it, you must call cpattern:Reference().

INPUTS

none

RESULTS

`pat`	the current source pattern (owned by Cairo)

5.38 ccontext:GetTarget

NAME

ccontext:GetTarget – get target surface

SYNOPSIS

```
surface = ccontext:GetTarget()
```

FUNCTION

Gets the target surface for the Cairo context as passed to cairo.Context().

This function will always return a valid handle, but the result can be a "nil" surface if the context is already in an error state. A `Nil` surface is indicated by csurface:Status() being different to `#CAIRO_STATUS_SUCCESS`.

This function returns the target surface. This object is owned by Cairo. To keep a reference to it, you must call csurface:Reference().

INPUTS

none

RESULTS

`surface`	the target surface (owned by Cairo)

5.39 ccontext:GetTolerance

NAME

ccontext:GetTolerance – get current tolerance value

SYNOPSIS

```
t = ccontext:GetTolerance()
```

FUNCTION

Gets the current tolerance value, as set by ccontext:SetTolerance().

INPUTS

none

RESULTS

`t`	the current tolerance value

5.40 ccontext:GlyphExtents

NAME

ccontext:GlyphExtents – get glyph extents

SYNOPSIS

```
t = ccontext:GlyphExtents(glyphs[, offset, num_glyphs])
```

FUNCTION

Gets the extents for a glyphs array. The extents describe a user-space rectangle that encloses the "inked" portion of the glyphs, as they would be drawn by ccontext:ShowGlyphs(). Additionally, the `XAdvance` and `YAdvance` values indicate the amount by which the current point would be advanced by ccontext:ShowGlyphs().

Note that whitespace glyphs do not contribute to the size of the rectangle (`Width` and `Height` fields).

This function returns a table describing the glyph extents. See ccontext:TextExtents() for a description of all table members.

INPUTS

`glyphs` a glyphs array

`offset` optional: offset into the array that specifies the starting glyph (defaults to 0 which means first glyph)

`num_glyphs` optional: number of glyphs to show (defaults to -1 which means all glyphs)

RESULTS

`t` table containing the glyphs extents

5.41 ccontext:GlyphPath

NAME

ccontext:GlyphPath – add glyphs to path

SYNOPSIS

```
ccontext:GlyphPath(glyphs[, offset, num_glyphs])
```

FUNCTION

Adds closed paths for the glyphs to the current path. The generated path if filled, achieves an effect similar to that of ccontext:ShowGlyphs().

INPUTS

`glyphs` a glyphs array

`offset` optional: offset into the array that specifies the starting glyph (defaults to 0 which means first glyph)

`num_glyphs` optional: number of glyphs to show (defaults to -1 which means all glyphs)

5.42 ccontext:GlyphStringPath

NAME

ccontext:GlyphStringPath – add glyphs to path

SYNOPSIS

```
ccontext:GlyphStringPath(font, glyphs)
```

FUNCTION

Adds the glyphs in `glyphs` to the current path in the specified Cairo context.

The origin of the glyphs (the left edge of the baseline) will be at the current point of the Cairo context.

INPUTS

`font`	a Pango font object
`glyphs`	a Pango glyph string object

5.43 ccontext:HasCurrentPoint

NAME

ccontext:HasCurrentPoint – has current point

SYNOPSIS

```
ok = ccontext:HasCurrentPoint()
```

FUNCTION

Returns whether a current point is defined on the current path. See ccontext:GetCurrentPoint() for details on the current point.

INPUTS

none

RESULTS

`ok`	whether a current point is defined

5.44 ccontext:IdentityMatrix

NAME

ccontext:IdentityMatrix – reset current transformation matrix

SYNOPSIS

```
ccontext:IdentityMatrix()
```

FUNCTION

Resets the current transformation matrix (CTM) by setting it equal to the identity matrix. That is, the user-space and device-space axes will be aligned and one user-space unit will transform to one device-space unit.

INPUTS

none

5.45 ccontext:InClip

NAME

ccontext:InClip – check if point is inside clip area

SYNOPSIS

```
ok = ccontext:InClip(x, y)
```

FUNCTION

Tests whether the given point is inside the area that would be visible through the current clip, i.e. the area that would be filled by a ccontext:Paint() operation.

See ccontext:Clip() and ccontext:ClipPreserve().

This function returns a non-zero value if the point is inside, or zero if outside.

INPUTS

`x`	X coordinate of the point to test
`y`	Y coordinate of the point to test

RESULTS

`ok`	a non-zero value if the point is inside, or zero if outside

5.46 ccontext:InFill

NAME

ccontext:InFill – check if point is inside fill area

SYNOPSIS

```
ok = ccontext:InFill(x, y)
```

FUNCTION

Tests whether the given point is inside the area that would be affected by a ccontext:Fill() operation given the current path and filling parameters. Surface dimensions and clipping are not taken into account.

See ccontext:Fill(), ccontext:SetFillRule() and ccontext:FillPreserve().

This function returns a non-zero value if the point is inside, or zero if outside.

INPUTS

`x`	X coordinate of the point to test
`y`	Y coordinate of the point to test

RESULTS

`ok`	a non-zero value if the point is inside, or zero if outside

5.47 ccontext:InStroke

NAME

ccontext:InStroke – check if point is inside stroke area

SYNOPSIS

```
ok = ccontext:InStroke(x, y)
```

FUNCTION

Tests whether the given point is inside the area that would be affected by a ccontext:Stroke() operation given the current path and stroking parameters. Surface dimensions and clipping are not taken into account.

See ccontext:Stroke(), ccontext:SetLineWidth(), ccontext:SetLineJoin(), ccontext:SetLineCap(), ccontext:SetDash(), and ccontext:StrokePreserve().

This function returns a non-zero value if the point is inside, or zero if outside.

INPUTS

`x`	X coordinate of the point to test
`y`	Y coordinate of the point to test

RESULTS

`ok`	a non-zero value if the point is inside, or zero if outside

5.48 ccontext:IsNull

NAME

ccontext:IsNull – check if context is invalid

SYNOPSIS

```
bool = ccontext:IsNull()
```

FUNCTION

Returns `True` if the context is `NULL`, i.e. invalid. If functions that allocate contexts fail, they might not throw an error but simply set the context to `NULL`. You can use this function to check if context allocation has failed in which case the context will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool`	`True` if the context is `NULL`, otherwise `False`

5.49 ccontext:LayoutLinePath

NAME
ccontext:LayoutLinePath – add text from layout line

SYNOPSIS
`ccontext:LayoutLinePath(line)`

FUNCTION
Adds the text from the Pango layout line specified by `line` to the current path in the specified Cairo context.

The origin of the glyphs (the left edge of the line) will be at the current point of the Cairo context.

INPUTS

`line` a Pango layout line object

5.50 ccontext:LayoutPath

NAME
ccontext:LayoutPath – add text from layout

SYNOPSIS
`ccontext:LayoutPath(layout)`

FUNCTION
Adds the text from the Pango layout specified by `layout` to the current path in the specified Cairo context.

The top-left corner of the Pango layout will be at the current point of the Cairo context.

INPUTS

`layout` a Pango layout object

5.51 ccontext:LineTo

NAME
ccontext:LineTo – add line to path

SYNOPSIS
`ccontext:LineTo(x, y)`

FUNCTION
Adds a line to the path from the current point to position (`x`, `y`) in user-space coordinates. After this call the current point will be (`x`, `y`).

If there is no current point before the call to ccontext:LineTo() this function will behave as ccontext:MoveTo() with `x` and `y` as the parameters.

INPUTS

`x` the X coordinate of the end of the new line

`y` the Y coordinate of the end of the new line

5.52 ccontext:Mask

NAME
ccontext:Mask – paint source with mask

SYNOPSIS
```
ccontext:Mask(pattern)
```

FUNCTION
A drawing operator that paints the current source using the alpha channel of `pattern` as a mask. Opaque areas of `pattern` are painted with the source, transparent areas are not painted.

INPUTS

`pattern` a Cairo pattern object

5.53 ccontext:MaskSurface

NAME
ccontext:MaskSurface – paint source with surface mask

SYNOPSIS
```
ccontext:MaskSurface(surface, surface_x, surface_y)
```

FUNCTION
A drawing operator that paints the current source using the alpha channel of `surface` as a mask. Opaque areas of `surface` are painted with the source, transparent areas are not painted.

INPUTS

`surface` a Cairo surface object

`surface_x`
X coordinate at which to place the origin of `surface`

`surface_y`
Y coordinate at which to place the origin of `surface`

5.54 ccontext:MoveTo

NAME
ccontext:MoveTo – begin new sub-path

SYNOPSIS
```
ccontext:MoveTo(x, y)
```

FUNCTION
Begin a new sub-path. After this call the current point will be (`x`, `y`).

INPUTS

`x`	the X coordinate of the new position
`y`	the Y coordinate of the new position

5.55 ccontext:NewPath

NAME
ccontext:NewPath – clear current path

SYNOPSIS
```
ccontext:NewPath()
```

FUNCTION
Clears the current path. After this call there will be no path and no current point.

INPUTS
none

5.56 ccontext:NewSubPath

NAME
ccontext:NewSubPath – begin new sub-path

SYNOPSIS
```
ccontext:NewSubPath()
```

FUNCTION
Begin a new sub-path. Note that the existing path is not affected. After this call there will be no current point.

In many cases, this call is not needed since new sub-paths are frequently started with ccontext:MoveTo().

A call to ccontext:NewSubPath() is particularly useful when beginning a new sub-path with one of the ccontext:Arc() calls. This makes things easier as it is no longer necessary to manually compute the arc's initial coordinates for a call to ccontext:MoveTo().

INPUTS
none

5.57 ccontext:Paint

NAME
ccontext:Paint – paint current source

SYNOPSIS
```
ccontext:Paint()
```

FUNCTION
A drawing operator that paints the current source everywhere within the current clip region.

INPUTS
none

5.58 ccontext:PaintWithAlpha

NAME
ccontext:PaintWithAlpha – paint current source with alpha

SYNOPSIS
```
ccontext:PaintWithAlpha(alpha)
```

FUNCTION
A drawing operator that paints the current source everywhere within the current clip region using a mask of constant alpha value `alpha`. The effect is similar to ccontext:Paint(), but the drawing is faded out using the alpha value.

INPUTS

`alpha`	alpha value, between 0 (transparent) and 1 (opaque)

5.59 ccontext:PangoContext

NAME
ccontext:PangoContext – create Pango context

SYNOPSIS
```
handle = ccontext:PangoContext()
```

FUNCTION
Creates a Pango context set up to match the current transformation and target surface of the Cairo context.

This context can then be used to create a layout using pango.Layout().

This function is a convenience function that creates a Pango context using the default font map, then updates it to the Cairo context. If you just need to create a layout for use with the Cairo context and do not need to access Pango context directly, you can use ccontext:PangoLayout() instead.

INPUTS
none

RESULTS

`handle`	the newly created Pango context

5.60 ccontext:PangoLayout

NAME

ccontext:PangoLayout – create Pango layout

SYNOPSIS

```
handle = ccontext:PangoLayout()
```

FUNCTION

Creates a Pango layout set up to match the current transformation and target surface of the Cairo context.

This layout can then be used for text measurement with functions like playout:GetSize() or drawing with functions like ccontext:ShowLayout(). If you change the transformation or target surface for the Cairo context, you need to call ccontext:UpdateLayout().

This function is the most convenient way to use Cairo with Pango, however it is slightly inefficient since it creates a separate Pango context object for each layout. This might matter in an application that was laying out large amounts of text.

INPUTS

none

RESULTS

`handle` the newly created Pango layout

5.61 ccontext:PathExtents

NAME

ccontext:PathExtents – get path extents

SYNOPSIS

```
x1, y1, x2, y2 = ccontext:PathExtents()
```

FUNCTION

Computes a bounding box in user-space coordinates covering the points on the current path. If the current path is empty, returns an empty rectangle ((0,0), (0,0)). Stroke parameters, fill rule, surface dimensions and clipping are not taken into account.

Contrast with ccontext:FillExtents() and ccontext:StrokeExtents() which return the extents of only the area that would be "inked" by the corresponding drawing operations.

The result of ccontext:PathExtents() is defined as equivalent to the limit of ccontext:StrokeExtents() with `#CAIRO_LINE_CAP_ROUND` as the line width approaches 0.0 (but never reaching the empty-rectangle returned by ccontext:StrokeExtents() for a line width of 0.0).

Specifically, this means that zero-area sub-paths such as ccontext:MoveTo();ccontext:LineTo() segments, (even degenerate cases where the coordinates to both calls are identical), will be considered as contributing to the extents. However, a lone ccontext:MoveTo() will not contribute to the results of ccontext:PathExtents().

INPUTS

none

RESULTS

`x1`	left of the resulting extents
`y1`	top of the resulting extents
`x2`	right of the resulting extents
`y2`	bottom of the resulting extents

5.62 ccontext:PopGroup

NAME

ccontext:PopGroup – pop group

SYNOPSIS

```
pat = ccontext:PopGroup()
```

FUNCTION

Terminates the redirection begun by a call to ccontext:PushGroup() or ccontext:PushGroupWithContent() and returns a new pattern containing the results of all drawing operations performed to the group.

The ccontext:PopGroup() function calls ccontext:Restore(), (balancing a call to ccontext:Save() by the push_group function), so that any changes to the graphics state will not be visible outside the group.

This function returns a newly created (surface) pattern containing the results of all drawing operations performed to the group. The caller owns the returned object and should call cpattern:Free() when finished with it.

INPUTS

none

RESULTS

`pat`	a newly created pattern

5.63 ccontext:PopGroupToSource

NAME

ccontext:PopGroupToSource – pop group to source

SYNOPSIS

```
ccontext:PopGroupToSource()
```

FUNCTION

Terminates the redirection begun by a call to ccontext:PushGroup() or ccontext:PushGroupWithContent() and installs the resulting pattern as the source pattern in the given Cairo context.

The behavior of this function is equivalent to the sequence of operations:

```
group = cr:PopGroup()
```

```
cr:SetSource(group)
group:Free()
```

but is more convenient as their is no need for a variable to store the short-lived handle to the pattern.

The ccontext:PopGroup() function calls ccontext:Restore(), (balancing a call to ccontext:Save() by the push group function), so that any changes to the graphics state will not be visible outside the group.

INPUTS

none

5.64 ccontext:PushGroup

NAME

ccontext:PushGroup – push group

SYNOPSIS

```
ccontext:PushGroup()
```

FUNCTION

Temporarily redirects drawing to an intermediate surface known as a group. The redirection lasts until the group is completed by a call to ccontext:PopGroup() or ccontext:PopGroupToSource(). These calls provide the result of any drawing to the group as a pattern, (either as an explicit object, or set as the source pattern).

This group functionality can be convenient for performing intermediate compositing. One common use of a group is to render objects as opaque within the group, (so that they occlude each other), and then blend the result with translucence onto the destination.

Groups can be nested arbitrarily deep by making balanced calls to ccontext:PushGroup() / ccontext:PopGroup(). Each call pushes/pops the new target group onto/from a stack.

The ccontext:PushGroup() function calls ccontext:Save() so that any changes to the graphics state will not be visible outside the group. The pop_group functions call ccontext:Restore().

By default the intermediate group will have a content type of `#CAIRO_CONTENT_COLOR_ALPHA`. Other content types can be chosen for the group by using ccontext:PushGroupWithContent() instead.

As an example, here is how one might fill and stroke a path with translucence, but without any portion of the fill being visible under the stroke:

```
cr:PushGroup()
cr:SetSource(fill_pattern)
cr:FillPreserve()
cr:SetSource(stroke_pattern)
cr:Stroke()
cr:PopGroupToSource()
cr:PaintWithAlpha(alpha)
```

INPUTS

none

5.65 ccontext:PushGroupWithContent

NAME

ccontext:PushGroupWithContent – push group with content

SYNOPSIS

```
ccontext:PushGroupWithContent(content)
```

FUNCTION

Temporarily redirects drawing to an intermediate surface known as a group. The redirection lasts until the group is completed by a call to ccontext:PopGroup() or ccontext:PopGroupToSource(). These calls provide the result of any drawing to the group as a pattern (either as an explicit object, or set as the source pattern).

The group will have a content type of `content`. The ability to control this content type is the only distinction between this function and ccontext:PushGroup() which you should see for a more detailed description of group rendering.

See csurface:GetContent() for a list of supported content types.

INPUTS

`content`	constant indicating the type of group that will be created (see above)

5.66 ccontext:Rectangle

NAME

ccontext:Rectangle – add rectangle to path

SYNOPSIS

```
ccontext:Rectangle(x, y, width, height)
```

FUNCTION

Adds a closed sub-path rectangle of the given size to the current path at position (`x`, `y`) in user-space coordinates.

This function is logically equivalent to:

```
cr:MoveTo(x, y)
cr:RelLineTo(width)
cr:RelLineTo(0, height)
cr:RelLineTo(-width, 0)
cr:ClosePath()
```

INPUTS

`x`	the X coordinate of the top left corner of the rectangle
`y`	the Y coordinate to the top left corner of the rectangle
`width`	the width of the rectangle
`height`	the height of the rectangle

5.67 ccontext:Reference

NAME

ccontext:Reference – increase reference count

SYNOPSIS

```
ccontext:Reference()
```

FUNCTION

Increases the reference count on the context by one. This prevents the context from being destroyed until a matching call to ccontext:Free() is made.

Use ccontext:GetReferenceCount() to get the number of references to a Cairo context.

INPUTS

none

5.68 ccontext:RelCurveTo

NAME

ccontext:RelCurveTo – add relative cubic Bezier spline

SYNOPSIS

```
ccontext:RelCurveTo(dx1, dy1, dx2, dy2, dx3, dy3)
```

FUNCTION

Relative-coordinate version of ccontext:CurveTo(). All offsets are relative to the current point. Adds a cubic Bezier spline to the path from the current point to a point offset from the current point by (`dx3`, `dy3`), using points offset by (`dx1`, `dy1`) and (`dx2`, `dy2`) as the control points. After this call the current point will be offset by (`dx3`, `dy3`).

Given a current point of (x, y), calling this function with `dx1`, `dy1`, `dx2`, `dy2`, `dx3`, `dy3` is logically equivalent to calling ccontext:CurveTo() with `x+dx1`, `y+dy1`, `x+dx2`, `y+dy2`, `x+dx3`, `y+dy3`.

It is an error to call this function with no current point. Doing so will cause the context to shutdown with a status of `#CAIRO_STATUS_NO_CURRENT_POINT`.

INPUTS

`dx1`	the X offset to the first control point
`dy1`	the Y offset to the first control point
`dx2`	the X offset to the second control point
`dy2`	the Y offset to the second control point
`dx3`	the X offset to the end of the curve
`dy3`	the Y offset to the end of the curve

5.69 ccontext:RelLineTo

NAME

ccontext:RelLineTo – add relative line

SYNOPSIS

```
ccontext:RelLineTo(dx, dy)
```

FUNCTION

Relative-coordinate version of ccontext:LineTo(). Adds a line to the path from the current point to a point that is offset from the current point by (`dx`, `dy`) in user space. After this call the current point will be offset by (`dx`, `dy`).

Given a current point of (x, y), calling this function with `dx`, `dy` is logically equivalent to calling ccontext:LineTo() with `x+dx` and `y+dy`.

It is an error to call this function with no current point. Doing so will cause the context to shutdown with a status of `#CAIRO_STATUS_NO_CURRENT_POINT`.

INPUTS

`dx`	the X offset to the end of the new line
`dy`	the Y offset to the end of the new line

5.70 ccontext:RelMoveTo

NAME

ccontext:RelMoveTo – begin new sub-path with relative point

SYNOPSIS

```
ccontext:RelMoveTo(dx, dy)
```

FUNCTION

Begin a new sub-path. After this call the current point will offset by (`x`, `y`).

Given a current point of (x, y), calling this function with `dx`, `dy` is logically equivalent to calling ccontext:MoveTo() with `x+dx` and `y+dy`.

It is an error to call this function with no current point. Doing so will cause the context to shutdown with a status of `#CAIRO_STATUS_NO_CURRENT_POINT`.

INPUTS

`dx`	the X offset
`dy`	the Y offset

5.71 ccontext:ResetClip

NAME

ccontext:ResetClip – reset current clip region

SYNOPSIS

```
ccontext:ResetClip()
```

FUNCTION

Reset the current clip region to its original, unrestricted state. That is, set the clip region to an infinitely large shape containing the target surface. Equivalently, if infinity is too hard to grasp, one can imagine the clip region being reset to the exact bounds of the target surface.

Note that code meant to be reusable should not call ccontext:ResetClip() as it will cause results unexpected by higher-level code which calls ccontext:Clip(). Consider using ccontext:Save() and ccontext:Restore() around ccontext:Clip() as a more robust means of temporarily restricting the clip region.

INPUTS

none

5.72 ccontext:Restore

NAME

ccontext:Restore – restore saved state

SYNOPSIS

```
ccontext:Restore()
```

FUNCTION

Restores the context to the state saved by a preceding call to ccontext:Save() and removes that state from the stack of saved states.

INPUTS

none

5.73 ccontext:Rotate

NAME

ccontext:Rotate – rotate current transformation matrix

SYNOPSIS

```
ccontext:Rotate(angle)
```

FUNCTION

Modifies the current transformation matrix (CTM) by rotating the user-space axes by `angle` radians. The rotation of the axes takes places after any existing transformation of user space. The rotation direction for positive angles is from the positive X axis toward the positive Y axis.

INPUTS

`angle`	angle (in radians) by which the user-space axes will be rotated

5.74 ccontext:Save

NAME
ccontext:Save – save current state

SYNOPSIS
`ccontext:Save()`

FUNCTION
Makes a copy of the current state of the context and saves it on an internal stack of saved states for the context. When ccontext:Restore() is called, the context will be restored to the saved state. Multiple calls to ccontext:Save() and ccontext:Restore() can be nested; each call to ccontext:Restore() restores the state from the matching paired ccontext:Save().

It isn't necessary to clear all saved states before a Cairo context is freed. If the reference count of a Cairo context drops to zero in response to a call to ccontext:Free(), any saved states will be freed along with the Cairo context.

INPUTS
none

5.75 ccontext:Scale

NAME
ccontext:Scale – scale current transformation matrix

SYNOPSIS
`ccontext:Scale(sx, sy)`

FUNCTION
Modifies the current transformation matrix (CTM) by scaling the X and Y user-space axes by `sx` and `sy` respectively. The scaling of the axes takes place after any existing transformation of user space.

INPUTS

`sx`	scale factor for the X dimension
`sy`	scale factor for the Y dimension

5.76 ccontext:SelectFontFace

NAME
ccontext:SelectFontFace – select font face

SYNOPSIS
`ccontext:SelectFontFace(family$, slant, weight)`

FUNCTION
Note: The ccontext:SelectFontFace() function call is part of what the Cairo designers call the "toy" text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications.

Selects a family and style of font from a simplified description as a family name, slant and weight. Cairo provides no operation to list available family names on the system (this is a "toy", remember), but the standard CSS2 generic family names, ("serif", "sans-serif", "cursive", "fantasy", "monospace"), are likely to work as expected.

If `family$` starts with the string "cairo:", or if no native font backends are compiled in, Cairo will use an internal font family. The internal font family recognizes many modifiers in the `family$` string, most notably, it recognizes the string "monospace". That is, the family name "cairo:monospace" will use the monospace version of the internal font family.

For "real" font selection, see the font-backend-specific font face creation functions for the font backend you are using, e.g. cairo.FontFace(). The resulting font face could then be used with cairo.ScaledFont() and ccontext:SetScaledFont().

Similarly, when using the "real" font support, you can call directly into the underlying font system, such as fontconfig or freetype, for operations such as listing available fonts, etc.

It is expected that most applications will need to use a more comprehensive font handling and text layout library, for example, pango), in conjunction with Cairo.

If text is drawn without a call to ccontext:SelectFontFace(), nor ccontext:SetFontFace() nor ccontext:SetScaledFont(), the default family is platform-specific, but is essentially "sans-serif".

The `slant` parameter can be one of the following constants:

```
#CAIRO_FONT_SLANT_NORMAL
#CAIRO_FONT_SLANT_ITALIC
#CAIRO_FONT_SLANT_OBLIQUE
```

The `weight` parameter can be one of the following constants:

```
#CAIRO_FONT_WEIGHT_NORMAL
#CAIRO_FONT_WEIGHT_BOLD
```

Default slant is `#CAIRO_FONT_SLANT_NORMAL`, and default weight is `#CAIRO_FONT_WEIGHT_NORMAL`.

This function is equivalent to a call to cairo.ToyFontFace() followed by ccontext:SetFontFace().

INPUTS

`family$`	a font family name
`slant`	the slant for the font (see above)
`weight`	the weight for the font (see above)

5.77 ccontext:SetAntialias

NAME

ccontext:SetAntialias – set antialias

SYNOPSIS

```
ccontext:SetAntialias(antialias)
```

FUNCTION

Set the antialiasing mode of the rasterizer used for drawing shapes. The `antialias` parameter can be one of the following constants:

`#CAIRO_ANTIALIAS_DEFAULT`
: Use the default antialiasing for the subsystem and target device.

`#CAIRO_ANTIALIAS_NONE`
: Use a bilevel alpha mask.

`#CAIRO_ANTIALIAS_GRAY`
: Perform single-color antialiasing (using shades of gray for black text on a white background, for example).

`#CAIRO_ANTIALIAS_SUBPIXEL`
: Perform antialiasing by taking advantage of the order of subpixel elements on devices such as LCD panels.

`#CAIRO_ANTIALIAS_FAST`
: Hint that the backend should perform some antialiasing but prefer speed over quality.

`#CAIRO_ANTIALIAS_GOOD`
: The backend should balance quality against performance.

`#CAIRO_ANTIALIAS_BEST`
: Hint that the backend should render at the highest quality, sacrificing speed if necessary.

Note that the `antialias` value is a hint, and a particular backend may or may not support a particular value. At the current time, no backend supports `#CAIRO_ANTIALIAS_SUBPIXEL` when drawing shapes.

Note that this option does not affect text rendering, instead see cfontoptions:SetAntialias().

INPUTS

`antialias`
: the new antialiasing mode (see above)

5.78 ccontext:SetDash

NAME

ccontext:SetDash – set dash pattern

SYNOPSIS

```
ccontext:SetDash(offset[, dash1, ...])
```

FUNCTION

Sets the dash pattern to be used by ccontext:Stroke(). A dash pattern is specified by a number of positive dash values starting with `dash1`. Each value provides the length of alternate "on" and "off" portions of the stroke. The `offset` specifies an offset into the pattern at which the stroke begins.

Each "on" segment will have caps applied as if the segment were a separate sub-path. In particular, it is valid to use an "on" length of 0.0 with `#CAIRO_LINE_CAP_ROUND` or `#CAIRO_LINE_CAP_SQUARE` in order to distributed dots or squares along a path.

Note: The length values are in user-space units as evaluated at the time of stroking. This is not necessarily the same as the user space at the time of ccontext:SetDash().

If you omit the optional `dash1` etc. arguments dashing is disabled.

If you pass just a single dash value, a symmetric pattern is assumed with alternating on and off portions of the size specified by the single value in `dash1`.

If any dash value is negative, or if all values are 0, then the Cairo context will be put into an error state with a status of `#CAIRO_STATUS_INVALID_DASH`.

INPUTS

`offset`	an offset into the dash pattern at which the stroke should start
`dash1`	optional: value for first stroke portion (on section)
`...`	optional: more values specifying alternate lengths of on and off stroke portions

5.79 ccontext:SetFillRule

NAME

ccontext:SetFillRule – set current fill rule

SYNOPSIS

```
ccontext:SetFillRule(fill_rule)
```

FUNCTION

Set the current fill rule within the Cairo context. The fill rule is used to determine which regions are inside or outside a complex (potentially self-intersecting) path. The current fill rule affects both ccontext:Fill() and ccontext:Clip(). The following values can be passed for `fill_rule`:

`#CAIRO_FILL_RULE_WINDING`
: If the path crosses the ray from left-to-right, counts +1. If the path crosses the ray from right to left, counts -1. Left and right are determined from the perspective of looking along the ray from the starting point. If the total count is non-zero, the point will be filled.

`#CAIRO_FILL_RULE_EVEN_ODD`
: Counts the total number of intersections, without regard to the orientation of the contour. If the total number of intersections is odd, the point will be filled.

The default fill rule is `#CAIRO_FILL_RULE_WINDING`.

INPUTS

`fill_rule`
: a fill rule (see above)

5.80 ccontext:SetFontFace

NAME
ccontext:SetFontFace – set font face

SYNOPSIS
`ccontext:SetFontFace(font_face)`

FUNCTION
Replaces the current font face object in the Cairo context with `font_face`. The replaced font face in the Cairo context will be destroyed if there are no other references to it.

INPUTS
`font_face`
a font face object or `Nil` to restore to the default font

5.81 ccontext:SetFontMatrix

NAME
ccontext:SetFontMatrix – set font matrix

SYNOPSIS
`ccontext:SetFontMatrix(matrix)`

FUNCTION
Sets the current font matrix to `matrix`. The font matrix gives a transformation from the design space of the font (in this space, the em-square is 1 unit by 1 unit) to user space. Normally, a simple scale is used (see ccontext:SetFontSize()), but a more complex font matrix can be used to shear the font or stretch it unequally along the two axes

INPUTS
`matrix` a matrix object describing a transform to be applied to the current font

5.82 ccontext:SetFontOptions

NAME
ccontext:SetFontOptions – set font options

SYNOPSIS
`ccontext:SetFontOptions(options)`

FUNCTION
Sets a set of custom font rendering options for the Cairo context. Rendering options are derived by merging these options with the options derived from underlying surface; if the value in `options` has a default value (like `#CAIRO_ANTIALIAS_DEFAULT`), then the value from the surface is used.

INPUTS
`options` font options object to use

5.83 ccontext:SetFontSize

NAME

ccontext:SetFontSize – set font size

SYNOPSIS

```
ccontext:SetFontSize(size)
```

FUNCTION

Sets the current font matrix to a scale by a factor of `size`, replacing any font matrix previously set with ccontext:SetFontSize() or ccontext:SetFontMatrix(). This results in a font size of `size` user space units. More precisely, this matrix will result in the font's em-square being a `size` by `size` square in user space.

If text is drawn without a call to ccontext:SetFontSize(), nor ccontext:SetFontMatrix() nor ccontext:SetScaledFont(), the default font size is 10.0.

INPUTS

`size` the new font size, in user space units

5.84 ccontext:SetLineCap

NAME

ccontext:SetLineCap – set line cap style

SYNOPSIS

```
ccontext:SetLineCap(line_cap)
```

FUNCTION

Sets the current line cap style within the Cairo context. The `line_cap` parameter can be one of the following constants:

`#CAIRO_LINE_CAP_BUTT`
: Start(stop) the line exactly at the start(end) point.

`#CAIRO_LINE_CAP_ROUND`
: Use a round ending, the center of the circle is the end point.

`#CAIRO_LINE_CAP_SQUARE`
: Use squared ending, the center of the square is the end point.

As with the other stroke parameters, the current line cap style is examined by ccontext:Stroke() and ccontext:StrokeExtents(), but does not have any effect during path construction.

The default line cap style is `#CAIRO_LINE_CAP_BUTT`.

INPUTS

`line_cap` a line cap style (see above)

5.85 ccontext:SetLineJoin

NAME

ccontext:SetLineJoin – set line join style

SYNOPSIS

```
ccontext:SetLineJoin(line_join)
```

FUNCTION

Sets the current line join style within the Cairo context. The `line_join` parameter can be one of the following constants:

`#CAIRO_LINE_JOIN_MITER`
: Use a sharp (angled) corner. See Section 5.88 [ccontext:SetMiterLimit], page 71, for details.

`#CAIRO_LINE_JOIN_ROUND`
: Use a rounded join, the center of the circle is the joint point.

`#CAIRO_LINE_JOIN_BEVEL`
: Use a cut-off join, the join is cut off at half the line width from the joint point.

As with the other stroke parameters, the current line join style is examined by ccontext:Stroke() and ccontext:StrokeExtents(), but does not have any effect during path construction.

The default line join style is `#CAIRO_LINE_JOIN_MITER`.

INPUTS

`line_join`
: a line join style (see above)

5.86 ccontext:SetLineWidth

NAME

ccontext:SetLineWidth – set line width

SYNOPSIS

```
ccontext:SetLineWidth(width)
```

FUNCTION

Sets the current line width within the Cairo context. The line width value specifies the diameter of a pen that is circular in user space, (though device-space pen may be an ellipse in general due to scaling/shear/rotation of the CTM).

Note: When the description above refers to user space and CTM it refers to the user space and CTM in effect at the time of the stroking operation, not the user space and CTM in effect at the time of the call to ccontext:SetLineWidth(). The simplest usage makes both of these spaces identical. That is, if there is no change to the CTM between a call to ccontext:SetLineWidth() and the stroking operation, then one can just pass user-space values to ccontext:SetLineWidth() and ignore this note.

As with the other stroke parameters, the current line width is examined by ccontext:Stroke() and ccontext:StrokeExtents(), but does not have any effect during path construction.

The default line width value is 2.0.

INPUTS

`width` a line width

5.87 ccontext:SetMatrix

NAME

ccontext:SetMatrix – set current transformation matrix

SYNOPSIS

```
ccontext:SetMatrix(matrix)
```

FUNCTION

Modifies the current transformation matrix (CTM) by setting it equal to `matrix`.

INPUTS

`matrix` a transformation matrix from user space to device space

5.88 ccontext:SetMiterLimit

NAME

ccontext:SetMiterLimit – set current miter limit

SYNOPSIS

```
ccontext:SetMiterLimit(limit)
```

FUNCTION

Sets the current miter limit within the Cairo context.

If the current line join style is set to `#CAIRO_LINE_JOIN_MITER` (see ccontext:SetLineJoin()), the miter limit is used to determine whether the lines should be joined with a bevel instead of a miter. Cairo divides the length of the miter by the line width. If the result is greater than the miter limit, the style is converted to a bevel.

As with the other stroke parameters, the current line miter limit is examined by ccontext:Stroke() and ccontext:StrokeExtents() but does not have any effect during path construction.

The default miter limit value is 10.0, which will convert joins with interior angles less than 11 degrees to bevels instead of miters. For reference, a miter limit of 2.0 makes the miter cutoff at 60 degrees, and a miter limit of 1.414 makes the cutoff at 90 degrees.

A miter limit for a desired angle can be computed as:

```
miterlimit = 1/sin(angle/2)
```

INPUTS

`limit` miter limit to set

5.89 ccontext:SetOperator

NAME

ccontext:SetOperator – set current compositing operator

SYNOPSIS

```
ccontext:SetOperator(op)
```

FUNCTION

Sets the compositing operator to be used for all drawing operations. The `op` parameter can be one of the following constants:

`#CAIRO_OPERATOR_CLEAR`
: Clear destination layer (bounded)

`#CAIRO_OPERATOR_SOURCE`
: Replace destination layer (bounded).

`#CAIRO_OPERATOR_OVER`
: Draw source layer on top of destination layer (bounded).

`#CAIRO_OPERATOR_IN`
: Draw source where there was destination content (unbounded).

`#CAIRO_OPERATOR_OUT`
: Draw source where there was no destination content (unbounded).

`#CAIRO_OPERATOR_ATOP`
: Draw source on top of destination content and only there.

`#CAIRO_OPERATOR_DEST`
: Ignore the source.

`#CAIRO_OPERATOR_DEST_OVER`
: Draw destination on top of source.

`#CAIRO_OPERATOR_DEST_IN`
: Leave destination only where there was source content (unbounded).

`#CAIRO_OPERATOR_DEST_OUT`
: Leave destination only where there was no source content.

`#CAIRO_OPERATOR_DEST_ATOP`
: Leave destination on top of source content and only there (unbounded).

`#CAIRO_OPERATOR_XOR`
: Source and destination are shown where there is only one of them.

`#CAIRO_OPERATOR_ADD`
: Source and destination layers are accumulated.

`#CAIRO_OPERATOR_SATURATE`
: Like over, but assuming source and dest are disjoint geometries.

`#CAIRO_OPERATOR_MULTIPLY`
: Source and destination layers are multiplied. This causes the result to be at least as dark as the darker inputs.

`#CAIRO_OPERATOR_SCREEN`
: Source and destination are complemented and multiplied. This causes the result to be at least as light as the lighter inputs.

`#CAIRO_OPERATOR_OVERLAY`
: Multiplies or screens, depending on the lightness of the destination color.

`#CAIRO_OPERATOR_DARKEN`
: Replaces the destination with the source if it is darker, otherwise keeps the source.

`#CAIRO_OPERATOR_LIGHTEN`
: Replaces the destination with the source if it is lighter, otherwise keeps the source.

`#CAIRO_OPERATOR_COLOR_DODGE`
: Brightens the destination color to reflect the source color.

`#CAIRO_OPERATOR_COLOR_BURN`
: Darkens the destination color to reflect the source color.

`#CAIRO_OPERATOR_HARD_LIGHT`
: Multiplies or screens, dependent on source color.

`#CAIRO_OPERATOR_SOFT_LIGHT`
: Darkens or lightens, dependent on source color.

`#CAIRO_OPERATOR_DIFFERENCE`
: Takes the difference of the source and destination color.

`#CAIRO_OPERATOR_EXCLUSION`
: Produces an effect similar to difference, but with lower contrast.

`#CAIRO_OPERATOR_HSL_HUE`
: Creates a color with the hue of the source and the saturation and luminosity of the target.

`#CAIRO_OPERATOR_HSL_SATURATION`
: Creates a color with the saturation of the source and the hue and luminosity of the target. Painting with this mode onto a gray area produces no change.

`#CAIRO_OPERATOR_HSL_COLOR`
: Creates a color with the hue and saturation of the source and the luminosity of the target. This preserves the gray levels of the target and is useful for coloring monochrome images or tinting color images.

`#CAIRO_OPERATOR_HSL_LUMINOSITY`
: Creates a color with the luminosity of the source and the hue and saturation of the target. This produces an inverse effect to `#CAIRO_OPERATOR_HSL_COLOR`.

The default operator is `#CAIRO_OPERATOR_OVER`.

INPUTS

`op` a compositing operator (see above)

5.90 ccontext:SetScaledFont

NAME

ccontext:SetScaledFont – set scaled font

SYNOPSIS

```
ccontext:SetScaledFont(scaled_font)
```

FUNCTION

Replaces the current font face, font matrix, and font options in the Cairo context with those of the `scaled_font`. Except for some translation, the current CTM of the Cairo context should be the same as that of the `scale_font`, which can be accessed using cscaledfont:GetCTM().

INPUTS

`scaled_font`
a scaled font object

5.91 ccontext:SetSource

NAME

ccontext:SetSource – set source pattern

SYNOPSIS

```
ccontext:SetSource(source)
```

FUNCTION

Sets the source pattern within the context to `source`. This pattern will then be used for any subsequent drawing operation until a new source pattern is set.

Note: The pattern's transformation matrix will be locked to the user space in effect at the time of ccontext:SetSource(). This means that further modifications of the current transformation matrix will not affect the source pattern. See cpattern:SetMatrix().

The default source pattern is a solid pattern that is opaque black, that is, it is equivalent to ccontext:SetSourceRGB() with the arguments all set to 0.0.

INPUTS

`source` a Cairo pattern object to be used as the source for subsequent drawing operations

5.92 ccontext:SetSourceRGB

NAME

ccontext:SetSourceRGB – set source RGB

SYNOPSIS

```
ccontext:SetSourceRGB(red, green, blue)
```

FUNCTION

Sets the source pattern within the context to an opaque color. This opaque color will then be used for any subsequent drawing operation until a new source pattern is set.

The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

The default source pattern is a solid pattern that is opaque black, that is, it is equivalent to ccontext:SetSourceRGB() with the arguments all set to 0.0.

INPUTS

`red`	red component of color
`green`	green component of color
`blue`	blue component of color

5.93 ccontext:SetSourceRGBA

NAME

ccontext:SetSourceRGBA – set source RGBA

SYNOPSIS

```
ccontext:SetSourceRGBA(red, green, blue, alpha)
```

FUNCTION

Sets the source pattern within the context to a translucent color. This color will then be used for any subsequent drawing operation until a new source pattern is set.

The color and alpha components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

Note that the color and alpha values are not premultiplied.

The default source pattern is a solid pattern that is opaque black, that is, it is equivalent to ccontext:SetSourceRGB() with the arguments all set to 0.0.

INPUTS

`red`	red component of color
`green`	green component of color
`blue`	blue component of color
`alpha`	alpha component of color

5.94 ccontext:SetSourceSurface

NAME

ccontext:SetSourceSurface – set source surface

SYNOPSIS

```
ccontext:SetSourceSurface(surface, x, y)
```

FUNCTION

This is a convenience function for creating a pattern from `surface` and setting it as the source in the context with ccontext:SetSource().

The `x` and `y` parameters give the user-space coordinate at which the surface origin should appear. The surface origin is its upper-left corner before any transformation has been applied. The `x` and `y` parameters are negated and then set as translation values in the pattern matrix.

Other than the initial translation pattern matrix, as described above, all other pattern attributes, such as its extend mode, are set to the default values as in cairo.PatternForSurface(). The resulting pattern can be queried with ccontext:GetSource() so that these attributes can be modified if desired, (e.g. to create a repeating pattern with cpattern:SetExtend()).

INPUTS

`surface`	a surface to be used to set the source pattern
`x`	User-space X coordinate for surface origin
`y`	User-space Y coordinate for surface origin

5.95 ccontext:SetTolerance

NAME

ccontext:SetTolerance – set tolerance

SYNOPSIS

```
ccontext:SetTolerance(tolerance)
```

FUNCTION

Sets the tolerance used when converting paths into trapezoids. Curved segments of the path will be subdivided until the maximum deviation between the original path and the polygonal approximation is less than `tolerance`. The default value is 0.1. A larger value will give better performance, a smaller value, better appearance. Reducing the value from the default value of 0.1 is unlikely to improve appearance significantly. The accuracy of paths within Cairo is limited by the precision of its internal arithmetic, and the prescribed `tolerance` is restricted to the smallest representable internal value.

INPUTS

`tolerance`
the tolerance, in device units (typically pixels)

5.96 ccontext:ShowErrorUnderline

NAME

ccontext:ShowErrorUnderline – draw error underline

SYNOPSIS

```
ccontext:ShowErrorUnderline(x, y, width, height)
```

FUNCTION

Draw a squiggly line in the specified Cairo context that approximately covers the given rectangle in the style of an underline used to indicate a spelling error.

The width of the underline is rounded to an integer number of up/down segments and the resulting rectangle is centered in the original rectangle.

INPUTS

`x` the X coordinate of one corner of the rectangle

`y` the Y coordinate of one corner of the rectangle

`width` non-negative width of the rectangle

`height` non-negative height of the rectangle

5.97 ccontext:ShowGlyphItem

NAME

ccontext:ShowGlyphItem – show glyph item

SYNOPSIS

```
ccontext:ShowGlyphItem(text$, glyph_item)
```

FUNCTION

Draws the glyphs in `glyph_item` in the specified Cairo context, embedding the text associated with the glyphs in the output if the output format supports it (PDF for example), otherwise it acts similar to ccontext:ShowGlyphString().

The origin of the glyphs (the left edge of the baseline) will be drawn at the current point of the Cairo context.

Note that `text` is the start of the text for layout, which is then indexed by the offset member of the glyph item.

INPUTS

`text$` the text that `glyph_item` refers to

`glyph_item`
a Pango glyph item object

5.98 ccontext:ShowGlyphs

NAME
ccontext:ShowGlyphs – show glyphs

SYNOPSIS
`ccontext:ShowGlyphs(glyphs[, offset, num_glyphs])`

FUNCTION
A drawing operator that generates the shape from a glyphs array, rendered according to the current font face, font size (font matrix), and font options.

INPUTS

`glyphs` glyphs array to show

`offset` optional: offset into the array that specifies the starting glyph (defaults to 0 which means first glyph)

`num_glyphs`
optional: number of glyphs to show (defaults to -1 which means all glyphs)

5.99 ccontext:ShowGlyphString

NAME
ccontext:ShowGlyphString – show glyph string

SYNOPSIS
`ccontext:ShowGlyphString(font, glyphs)`

FUNCTION
Draws the glyphs in `glyphs` in the specified Cairo context.

The origin of the glyphs (the left edge of the baseline) will be drawn at the current point of the Cairo context.

INPUTS

`font` a Pango font object

`glyphs` a Pango glyph string object

5.100 ccontext:ShowLayout

NAME
ccontext:ShowLayout – show layout

SYNOPSIS
`ccontext:ShowLayout(layout)`

FUNCTION
Draws a Pango layout in the specified Cairo context.

The top-left corner of the Pango layout will be drawn at the current point of the Cairo context.

INPUTS

`layout` a Pango layout object

5.101 ccontext:ShowLayoutLine

NAME

ccontext:ShowLayoutLine – show layout line

SYNOPSIS

```
ccontext:ShowLayoutLine(line)
```

FUNCTION

Draws a Pango layout line in the specified Cairo context.

The origin of the glyphs (the left edge of the line) will be drawn at the current point of the Cairo context.

INPUTS

`line` a Pango layout line object

5.102 ccontext:ShowPage

NAME

ccontext:ShowPage – show page

SYNOPSIS

```
ccontext:ShowPage()
```

FUNCTION

Emits and clears the current page for backends that support multiple pages. Use ccontext:CopyPage() if you don't want to clear the page.

This is a convenience function that simply calls csurface:ShowPage() on `cr's` target.

INPUTS

none

5.103 ccontext:ShowText

NAME

ccontext:ShowText – show text

SYNOPSIS

```
ccontext:ShowText(s$)
```

FUNCTION

A drawing operator that generates the shape from the string passed in `s$`, rendered according to the current font face, font size (font matrix), and font options.

This function first computes a set of glyphs for the string of text. The first glyph is placed so that its origin is at the current point. The origin of each subsequent glyph is offset from that of the previous glyph by the advance values of the previous glyph.

After this call the current point is moved to the origin of where the next glyph would be placed in this same progression. That is, the current point will be at the origin of the final glyph offset by its advance values. This allows for easy display of a single logical string with multiple calls to ccontext:ShowText().

Note: The ccontext:ShowText() function call is part of what the Cairo designers call the "toy" text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications. See ccontext:ShowGlyphs() for the "real" text display API in Cairo.

INPUTS

`s$` string of text to show

5.104 ccontext:Status

NAME

ccontext:Status – get context status

SYNOPSIS

```
status = ccontext:Status()
```

FUNCTION

Checks whether an error has previously occurred for this context. The following status codes are currently defined:

`#CAIRO_STATUS_SUCCESS`
: No error has occurred.

`#CAIRO_STATUS_NO_MEMORY`
: Out of memory

`#CAIRO_STATUS_INVALID_RESTORE`
: ccontext:Restore() called without matching ccontext:Save()

`#CAIRO_STATUS_INVALID_POP_GROUP`
: No saved group to pop, i.e. ccontext:PopGroup() without matching ccontext:PushGroup()

`#CAIRO_STATUS_NO_CURRENT_POINT`
: No current point defined

`#CAIRO_STATUS_INVALID_MATRIX`
: Invalid matrix (not invertible)

`#CAIRO_STATUS_INVALID_STATUS`
: Invalid value for an input Cairo status

`#CAIRO_STATUS_NULL_POINTER`
: `NULL` pointer

`#CAIRO_STATUS_INVALID_STRING`
: Input string not valid UTF-8

`#CAIRO_STATUS_INVALID_PATH_DATA`
: Input path data not valid

`#CAIRO_STATUS_READ_ERROR`
: Error while reading from input stream

`#CAIRO_STATUS_WRITE_ERROR`
: Error while writing to output stream

`#CAIRO_STATUS_SURFACE_FINISHED`
: Target surface has been finished

`#CAIRO_STATUS_SURFACE_TYPE_MISMATCH`
: The surface type is not appropriate for the operation

`#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`
: The pattern type is not appropriate for the operation

`#CAIRO_STATUS_INVALID_CONTENT`
: Invalid value for an input Cairo content

`#CAIRO_STATUS_INVALID_FORMAT`
: Invalid value for an input Cairo format

`#CAIRO_STATUS_INVALID_VISUAL`
: Invalid value for an input visual

`#CAIRO_STATUS_FILE_NOT_FOUND`
: File not found

`#CAIRO_STATUS_INVALID_DASH`
: Invalid value for a dash setting

`#CAIRO_STATUS_INVALID_DSC_COMMENT`
: Invalid value for a DSC comment

`#CAIRO_STATUS_INVALID_INDEX`
: Invalid index passed to getter

`#CAIRO_STATUS_CLIP_NOT_REPRESENTABLE`
: Clip region not representable in desired format

`#CAIRO_STATUS_TEMP_FILE_ERROR`
: Error creating or writing to a temporary file

`#CAIRO_STATUS_INVALID_STRIDE`
: Invalid value for stride

`#CAIRO_STATUS_FONT_TYPE_MISMATCH`
: The font type is not appropriate for the operation

`#CAIRO_STATUS_USER_FONT_IMMUTABLE`
: The user-font is immutable

`#CAIRO_STATUS_USER_FONT_ERROR`
: Error occurred in a user-font callback function

`#CAIRO_STATUS_NEGATIVE_COUNT`
: Negative number used where it is not allowed

`#CAIRO_STATUS_INVALID_CLUSTERS`
: Input clusters do not represent the accompanying text and glyph array

`#CAIRO_STATUS_INVALID_SLANT`
: Invalid value for an input Cairo font slant

`#CAIRO_STATUS_INVALID_WEIGHT`
: Invalid value for an input Cairo font weight

`#CAIRO_STATUS_INVALID_SIZE`
: Invalid value (typically too big) for the size of the input (surface, pattern, etc.)

`#CAIRO_STATUS_USER_FONT_NOT_IMPLEMENTED`
: User-font method not implemented

`#CAIRO_STATUS_DEVICE_TYPE_MISMATCH`
: The device type is not appropriate for the operation

`#CAIRO_STATUS_DEVICE_ERROR`
: An operation to the device caused an unspecified error

`#CAIRO_STATUS_INVALID_MESH_CONSTRUCTION`
: A mesh pattern construction operation was used outside begin end patch pair

`#CAIRO_STATUS_DEVICE_FINISHED`
: Target device has been finished

`#CAIRO_STATUS_JBIG2_GLOBAL_MISSING`
: `#CAIRO_MIME_TYPE_JBIG2_GLOBAL_ID` has been used on at least one image but no image provided `#CAIRO_MIME_TYPE_JBIG2_GLOBAL`

`#CAIRO_STATUS_PNG_ERROR`
: Error occurred in libpng while reading from or writing to a PNG file

`#CAIRO_STATUS_FREETYPE_ERROR`
: Error occurred in libfreetype

`#CAIRO_STATUS_WIN32_GDI_ERROR`
: Error occurred in the Windows Graphics Device Interface

`#CAIRO_STATUS_TAG_ERROR`
: Invalid tag name, attributes, or nesting

`#CAIRO_STATUS_DWRITE_ERROR`
: Error occurred in the Windows Direct Write API

`#CAIRO_STATUS_SVG_FONT_ERROR`
: Error occurred in OpenType-SVG font rendering

INPUTS
none

RESULTS
`status` the current status of this context

5.105 ccontext:Stroke

NAME
ccontext:Stroke – stroke current path

SYNOPSIS
```
ccontext:Stroke()
```

FUNCTION
A drawing operator that strokes the current path according to the current line width, line join, line cap, and dash settings. After ccontext:Stroke(), the current path will be cleared from the Cairo context. See ccontext:SetLineWidth(), ccontext:SetLineJoin(), ccontext:SetLineCap(), ccontext:SetDash(), and ccontext:StrokePreserve().

Note: Degenerate segments and sub-paths are treated specially and provide a useful result. These can result in two different situations:

1. Zero-length "on" segments set in ccontext:SetDash(). If the cap style is `#CAIRO_LINE_CAP_ROUND` or `#CAIRO_LINE_CAP_SQUARE` then these segments will be drawn as circular dots or squares respectively. In the case of `#CAIRO_LINE_CAP_SQUARE`, the orientation of the squares is determined by the direction of the underlying path.
2. A sub-path created by ccontext:MoveTo() followed by either a ccontext:ClosePath() or one or more calls to ccontext:LineTo() to the same coordinate as the ccontext:MoveTo(). If the cap style is `#CAIRO_LINE_CAP_ROUND` then these sub-paths will be drawn as circular dots. Note that in the case of `#CAIRO_LINE_CAP_SQUARE` a degenerate sub-path will not be drawn at all, (since the correct orientation is indeterminate).

In no case will a cap style of `#CAIRO_LINE_CAP_BUTT` cause anything to be drawn in the case of either degenerate segments or sub-paths.

INPUTS
none

5.106 ccontext:StrokeExtents

NAME
ccontext:StrokeExtents – get stroke extents

SYNOPSIS
```
x1, y1, x2, y2 = ccontext:StrokeExtents()
```

FUNCTION
Computes a bounding box in user coordinates covering the area that would be affected, the "inked" area, by a ccontext:Stroke() operation given the current path and stroke

parameters. If the current path is empty, returns an empty rectangle ((0,0), (0,0)). Surface dimensions and clipping are not taken into account.

Note that if the line width is set to exactly zero, then ccontext:StrokeExtents() will return an empty rectangle. Contrast with ccontext:PathExtents() which can be used to compute the non-empty bounds as the line width approaches zero.

Note that ccontext:StrokeExtents() must necessarily do more work to compute the precise inked areas in light of the stroke parameters, so ccontext:PathExtents() may be more desirable for sake of performance if non-inked path extents are desired.

See ccontext:Stroke(), ccontext:SetLineWidth(), ccontext:SetLineJoin(), ccontext:SetLineCap(), ccontext:SetDash(), and ccontext:StrokePreserve().

INPUTS

none

RESULTS

`x1`	left of the resulting extents
`y1`	top of the resulting extents
`x2`	right of the resulting extents
`y2`	bottom of the resulting extents

5.107 ccontext:StrokePreserve

NAME

ccontext:StrokePreserve – stroke path and preserve it

SYNOPSIS

```
ccontext:StrokePreserve()
```

FUNCTION

A drawing operator that strokes the current path according to the current line width, line join, line cap, and dash settings. Unlike ccontext:Stroke(), ccontext:StrokePreserve() preserves the path within the Cairo context.

See ccontext:SetLineWidth(), ccontext:SetLineJoin(), ccontext:SetLineCap(), ccontext:SetDash(), and ccontext:StrokePreserve().

INPUTS

none

5.108 ccontext:TagBegin

NAME

ccontext:TagBegin – mark beginning of a tag

SYNOPSIS

```
ccontext:TagBegin(tag_name$, attributes$)
```

FUNCTION

Marks the beginning of the `tag_name$` structure. Call ccontext:TagEnd() with the same `tag_name$` to mark the end of the structure. The `tag_name$` parameter can be one of the following constants:

`#CAIRO_TAG_DEST`
: Create a destination for a hyperlink.

`#CAIRO_TAG_LINK`
: Create hyperlink.

The `attributes$` string is of the form "key1=value2 key2=value2 ...". Values may be boolean (true/false or 1/0), integer, float, string, or an array.

String values are enclosed in single quotes ('). Single quotes and backslashes inside the string should be escaped with a backslash.

Boolean values may be set to true by only specifying the key. eg the attribute string "key" is the equivalent to "key=true".

Arrays are enclosed in '[]'. eg "rect=[1.2 4.3 2.0 3.0]".

If no attributes are required, `attributes` can be an empty string.

See the tags and links description in the Cairo manual for the list of tags and attributes.

Invalid nesting of tags or invalid attributes will cause the context to shutdown with a status of `#CAIRO_STATUS_TAG_ERROR`.

See ccontext:TagEnd().

INPUTS

`tag_name$`
: tag name (see above)

`attributes$`
: tag attributes (see above)

EXAMPLE

```
cr:TagBegin(#CAIRO_TAG_LINK, "dest='mydest' internal")
cr:MoveTo(50, 50)
cr:ShowText("This is a hyperlink.")
cr:TagEnd(#CAIRO_TAG_LINK)
```

The code above creates a hyperlink.

5.109 ccontext:TagEnd

NAME

ccontext:TagEnd – mark end of a tag

SYNOPSIS

```
ccontext:TagEnd(tag_name$)
```

FUNCTION

Marks the end of the `tag_name$` structure. See ccontext:TagBegin() for a list of tag names.

Invalid nesting of tags will cause the context to shutdown with a status of `#CAIRO_STATUS_TAG_ERROR`.

INPUTS

`tag_name$`
: tag name

5.110 ccontext:TextExtents

NAME

ccontext:TextExtents – get text extents

SYNOPSIS

```
t = ccontext:TextExtents(s$)
```

FUNCTION

Gets the extents for a string of text. The extents describe a user-space rectangle that encloses the "inked" portion of the text, (as it would be drawn by ccontext:ShowText()). Additionally, the `XAdvance` and `YAdvance` values indicate the amount by which the current point would be advanced by ccontext:ShowText().

This function returns a table describing the text extents. The table contains the following fields:

`XBearing`
: The horizontal distance from the origin to the leftmost part of the glyphs as drawn. Positive if the glyphs lie entirely to the right of the origin.

`YBearing`
: The vertical distance from the origin to the topmost part of the glyphs as drawn. Positive only if the glyphs lie completely below the origin; will usually be negative.

`Width`
: Width of the glyphs as drawn.

`Height`
: Height of the glyphs as drawn.

`XAdvance`
: Distance to advance in the X direction after drawing these glyphs.

`YAdvance`
: Distance to advance in the Y direction after drawing these glyphs. Will typically be zero except for vertical text layout as found in East-Asian languages.

Note that whitespace characters do not directly contribute to the size of the rectangle (`Width` and `Height`). They do contribute indirectly by changing the position of non-whitespace characters. In particular, trailing whitespace characters are likely to not affect the size of the rectangle, though they will affect the `XAdvance` and `YAdvance` values.

Also note that because text extents are in user-space coordinates, they are mostly, but not entirely, independent of the current transformation matrix. If you call ccontext:Scale() with scaling coefficients of 2.0 on each axis, the text will be drawn twice as big, but the reported text extents will not be doubled. They will change slightly due to hinting (so you can't assume that metrics are independent of the transformation matrix), but otherwise will remain unchanged.

INPUTS

`s$` string of text

RESULTS

`t` table containing the text extents (see above)

5.111 ccontext:TextPath

NAME

ccontext:TextPath – add text to path

SYNOPSIS

```
ccontext:TextPath(s$)
```

FUNCTION

Adds closed paths for text to the current path. The generated path if filled, achieves an effect similar to that of ccontext:ShowText().

Text conversion and positioning is done similar to ccontext:ShowText().

Like ccontext:ShowText(), after this call the current point is moved to the origin of where the next glyph would be placed in this same progression. That is, the current point will be at the origin of the final glyph offset by its advance values. This allows for chaining multiple calls to to ccontext:TextPath() without having to set current point in between.

Note: The ccontext:TextPath() function call is part of what the Cairo designers call the "toy" text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications. See ccontext:GlyphPath() for the "real" text path API in Cairo.

INPUTS

`s$` string of text

5.112 ccontext:Transform

NAME

ccontext:Transform – modify current transformation matrix

SYNOPSIS

```
ccontext:Transform(matrix)
```

FUNCTION

Modifies the current transformation matrix (CTM) by applying `matrix` as an additional transformation. The new transformation of user space takes place after any existing transformation.

INPUTS

`matrix` a transformation matrix to be applied to the user-space axes

5.113 ccontext:Translate

NAME
ccontext:Translate – translate current transformation matrix

SYNOPSIS
```
ccontext:Translate(tx, ty)
```

FUNCTION
Modifies the current transformation matrix (CTM) by translating the user-space origin by (`tx`, `ty`). This offset is interpreted as a user-space coordinate according to the CTM in place before the new call to ccontext:Translate(). In other words, the translation of the user-space origin takes place after any existing transformation.

INPUTS

`tx`	amount to translate in the X direction
`ty`	amount to translate in the Y direction

5.114 ccontext:UpdateLayout

NAME
ccontext:UpdateLayout – update Pango layout

SYNOPSIS
```
ccontext:UpdateLayout(layout)
```

FUNCTION
Updates the private Pango context of a Pango layout created with ccontext:PangoLayout() to match the current transformation and target surface of a Cairo context.

INPUTS

`layout`	a Pango layout from ccontext:PangoLayout()

5.115 ccontext:UserToDevice

NAME
ccontext:UserToDevice – user to device

SYNOPSIS
```
dx, dy = ccontext:UserToDevice(ux, uy)
```

FUNCTION
Transform a coordinate from user space to device space by multiplying the given point by the current transformation matrix (CTM).

INPUTS

`ux`	X value of coordinate (user space)
`uy`	Y value of coordinate (user space)

RESULTS

`dx`	X value of coordinate (device space)
`dy`	Y value of coordinate (device space)

5.116 ccontext:UserToDeviceDistance

NAME

ccontext:UserToDeviceDistance – user to device distance

SYNOPSIS

```
dx, dy = ccontext:UserToDeviceDistance(ux, uy)
```

FUNCTION

Transform a distance vector from user space to device space. This function is similar to ccontext:UserToDevice() except that the translation components of the CTM will be ignored when transforming (`dx`, `dy`).

INPUTS

`ux`	X component of a distance vector (user space)
`uy`	Y component of a distance vector (user space)

RESULTS

`dx`	X component of a distance vector (device space)
`dy`	Y component of a distance vector (device space)

6 Cairo font face

6.1 cfontface:Free

NAME
cfontface:Free – free font face

SYNOPSIS
`cfontface:Free()`

FUNCTION
Decreases the reference count on the font face by one. If the result is zero, then the font face and all associated resources are freed. See cfontface:Reference().

INPUTS
none

6.2 cfontface:GetFamily

NAME
cfontface:GetFamily – get family name

SYNOPSIS
`f$ = cfontface:GetFamily()`

FUNCTION
Gets the family name of a toy font.

INPUTS
none

RESULTS

`f$`	the family name of the font

6.3 cfontface:GetReferenceCount

NAME
cfontface:GetReferenceCount – get reference count

SYNOPSIS
`count = cfontface:GetReferenceCount()`

FUNCTION
Returns the current reference count of the font face.

INPUTS
none

RESULTS

`count`	the current reference count of the font face

6.4 cfontface:GetSlant

NAME
cfontface:GetSlant – get font slant

SYNOPSIS
```
slant = cfontface:GetSlant()
```

FUNCTION
Gets the slant a toy font. This will be one of the following constants:

```
#CAIRO_FONT_SLANT_NORMAL
#CAIRO_FONT_SLANT_ITALIC
#CAIRO_FONT_SLANT_OBLIQUE
```

INPUTS
none

RESULTS
`slant` the slant value (see above)

6.5 cfontface:GetType

NAME
cfontface:GetType – get font backend

SYNOPSIS
```
type = cfontface:GetType()
```

FUNCTION
This function returns the type of the backend used to create a font face. This will be one of the following constants:

`#CAIRO_FONT_TYPE_TOY`
The font was created using cairo's toy font api.

`#CAIRO_FONT_TYPE_FT:`
The font is of type FreeType.

`#CAIRO_FONT_TYPE_WIN32`
The font is of type Win32.

`#CAIRO_FONT_TYPE_QUARTZ`
The font is of type Quartz.

`#CAIRO_FONT_TYPE_USER`
The font was create using cairo's user font api.

`#CAIRO_FONT_TYPE_DWRITE`
The font is of type Win32 DWrite.

INPUTS
none

RESULTS

`type` the font face type (see above)

6.6 cfontface:GetWeight

NAME

cfontface:GetWeight – get font weight

SYNOPSIS

```
weight = cfontface:GetWeight()
```

FUNCTION

Gets the weight a toy font. This will be one of the following constants:

```
#CAIRO_FONT_WEIGHT_NORMAL
#CAIRO_FONT_WEIGHT_BOLD
```

INPUTS

none

RESULTS

`weight` the weight value (see above)

6.7 cfontface:IsNull

NAME

cfontface:IsNull – check if font face is invalid

SYNOPSIS

```
bool = cfontface:IsNull()
```

FUNCTION

Returns `True` if the font face is `NULL`, i.e. invalid. If functions that allocate font faces fail, they might not throw an error but simply set the font face to `NULL`. You can use this function to check if font face allocation has failed in which case the font face will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the font face is `NULL`, otherwise `False`

6.8 cfontface:Reference

NAME
cfontface:Reference – increase reference count

SYNOPSIS
```
cfontface:Reference()
```

FUNCTION
Increases the reference count on the font face by one. This prevents the font face from being destroyed until a matching call to cfontface:Free() is made.

Use cfontface:GetReferenceCount() to get the number of references to a Cairo font face.

INPUTS
none

6.9 cfontface:Status

NAME
cfontface:Status – get font face status

SYNOPSIS
```
status = cfontface:Status()
```

FUNCTION
Checks whether an error has previously occurred for this font face.

This function returns `#CAIRO_STATUS_SUCCESS` or another error such as `#CAIRO_STATUS_NO_MEMORY`.

INPUTS
none

RESULTS
`status` a Cairo status code

7 Cairo font options

7.1 cfontoptions:Copy

NAME

cfontoptions:Copy – copy font options object

SYNOPSIS

```
handle = cfontoptions:Copy()
```

FUNCTION

Allocates a new font options object copying the option values from the original.

This function returns a newly allocated Cairo font options object. Free with cfontoptions:Free(). This function always returns a valid handle; if memory cannot be allocated, then a special error object is returned where all operations on the object do nothing. You can check for this with cfontoptions:Status().

INPUTS

none

RESULTS

`handle` a newly allocated Cairo font options object

7.2 cfontoptions:Equal

NAME

cfontoptions:Equal – check for equality

SYNOPSIS

```
ok = cfontoptions:Equal(other)
```

FUNCTION

Compares two font options objects for equality.

This function returns `True` if all fields of the two font options objects match. Note that this function will return `False` if either object is in error.

INPUTS

`other` another Cairo font options object

RESULTS

`ok` `True` if all fields of the two font options objects match

7.3 cfontoptions:Free

NAME

cfontoptions:Free – free font options object

SYNOPSIS
`cfontoptions:Free()`

FUNCTION
Destroys a Cairo font options object object created with cairo.FontOptions() or cfontoptions:Copy().

INPUTS
none

7.4 cfontoptions:GetAntialias

NAME
cfontoptions:GetAntialias – get antialias mode

SYNOPSIS
`mode = cfontoptions:GetAntialias()`

FUNCTION
Gets the antialiasing mode for the font options object. See ccontext:SetAntialias() for a list of antialiasing modes.

INPUTS
none

RESULTS

`mode`	the antialiasing mode

7.5 cfontoptions:GetHintMetrics

NAME
cfontoptions:GetHintMetrics – get hint metrics

SYNOPSIS
`metrics = cfontoptions:GetHintMetrics()`

FUNCTION
Gets the metrics hinting mode for the font options object. See cfontoptions:SetHintMetrics() for a list of hint metrics.

INPUTS
none

RESULTS

`metrics`	the metrics hinting mode for the font options object

7.6 cfontoptions:GetHintStyle

NAME
cfontoptions:GetHintStyle – get hint style

SYNOPSIS
```
style = cfontoptions:GetHintStyle()
```

FUNCTION
Gets the hint style for font outlines for the font options object. See cfontoptions:SetHintStyle() for a list of hint styles.

INPUTS
none

RESULTS
`style` the hint style for the font options object

7.7 cfontoptions:GetSubpixelOrder

NAME
cfontoptions:GetSubpixelOrder – get subpixel order

SYNOPSIS
```
order = cfontoptions:GetSubpixelOrder()
```

FUNCTION
Gets the subpixel order for the font options object. See cfontoptions:SetSubpixelOrder() for a list of subpixel orders.

INPUTS
none

RESULTS
`order` the subpixel order for the font options object

7.8 cfontoptions:GetVariations

NAME
cfontoptions:GetVariations – get font variations

SYNOPSIS
```
v$ = cfontoptions:GetVariations()
```

FUNCTION
Gets the OpenType font variations for the font options object. See cfontoptions:SetVariations() for details about the string format.

INPUTS
none

RESULTS

`v$` the font variations for the font options object

7.9 cfontoptions:Hash

NAME

cfontoptions:Hash – compute object hash

SYNOPSIS

```
v = cfontoptions:Hash()
```

FUNCTION

Compute a hash for the font options object; this value will be useful when storing an object containing a Cairo font options object in a hash table.

This function returns the hash value for the font options object. The return value can be cast to a 32-bit type if a 32-bit hash value is needed.

INPUTS

none

RESULTS

`v` the hash value for the font options object

7.10 cfontoptions:IsNull

NAME

cfontoptions:IsNull – check if font options object is invalid

SYNOPSIS

```
bool = cfontoptions:IsNull()
```

FUNCTION

Returns `True` if the font options object is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the object will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the object is `NULL`, otherwise `False`

7.11 cfontoptions:Merge

NAME
cfontoptions:Merge – merge font options

SYNOPSIS
`cfontoptions:Merge(other)`

FUNCTION
Merges non-default options from `other` into the font options object, replacing existing values. This operation can be thought of as somewhat similar to compositing `other` onto the font options object with the operation of `#CAIRO_OPERATOR_OVER`.

INPUTS
`other` another Cairo font options object

7.12 cfontoptions:SetAntialias

NAME
cfontoptions:SetAntialias – set antialias mode

SYNOPSIS
`cfontoptions:SetAntialias(antialias)`

FUNCTION
Sets the antialiasing mode for the font options object. This specifies the type of antialiasing to do when rendering text. See ccontext:SetAntialias() for a list of antialiasing modes.

INPUTS
`antialias`
the new antialiasing mode

7.13 cfontoptions:SetHintMetrics

NAME
cfontoptions:SetHintMetrics – set hint metrics

SYNOPSIS
`cfontoptions:SetHintMetrics(hint_metrics)`

FUNCTION
Sets the metrics hinting mode for the font options object. This controls whether metrics are quantized to integer values in device units.

The following constants can be passed in the `hint_metrics` argument:

`#CAIRO_HINT_METRICS_DEFAULT`
Hint metrics in the default manner for the font backend and target device.

`#CAIRO_HINT_METRICS_OFF`
Do not hint font metrics.

`#CAIRO_HINT_METRICS_ON`
: Hint font metrics.

INPUTS

`hint_metrics`
: the new metrics hinting mode

7.14 cfontoptions:SetHintStyle

NAME
cfontoptions:SetHintStyle – set hint style

SYNOPSIS
`cfontoptions:SetHintStyle(hint_style)`

FUNCTION
Sets the hint style for font outlines for the font options object. This controls whether to fit font outlines to the pixel grid, and if so, whether to optimize for fidelity or contrast.

The following constants can be passed in the `hint_style` argument:

`#CAIRO_HINT_STYLE_DEFAULT`
: Use the default hint style for font backend and target device.

`#CAIRO_HINT_STYLE_NONE`
: Do not hint outlines.

`#CAIRO_HINT_STYLE_SLIGHT`
: Hint outlines slightly to improve contrast while retaining good fidelity to the original shapes.

`#CAIRO_HINT_STYLE_MEDIUM`
: Hint outlines with medium strength giving a compromise between fidelity to the original shapes and contrast.

`#CAIRO_HINT_STYLE_FULL`
: Hint outlines to maximize contrast.

INPUTS

`hint_style`
: the new hint style

7.15 cfontoptions:SetSubpixelOrder

NAME
cfontoptions:SetSubpixelOrder – set subpixel order

SYNOPSIS
`cfontoptions:SetSubpixelOrder(subpixel_order)`

FUNCTION

Sets the subpixel order for the font options object. The subpixel order specifies the order of color elements within each pixel on the display device when rendering with an antialiasing mode of `#CAIRO_ANTIALIAS_SUBPIXEL`.

The following constants can be passed in the `subpixel_order` argument:

`#CAIRO_SUBPIXEL_ORDER_DEFAULT`
: Use the default subpixel order for for the target device.

`#CAIRO_SUBPIXEL_ORDER_RGB`
: Subpixel elements are arranged horizontally with red at the left.

`#CAIRO_SUBPIXEL_ORDER_BGR`
: Subpixel elements are arranged horizontally with blue at the left.

`#CAIRO_SUBPIXEL_ORDER_VRGB`
: Subpixel elements are arranged vertically with red at the top.

`#CAIRO_SUBPIXEL_ORDER_VBGR`
: Subpixel elements are arranged vertically with blue at the top.

INPUTS

`subpixel_order`
: the new subpixel order

7.16 cfontoptions:SetVariations

NAME

cfontoptions:SetVariations – set variations

SYNOPSIS

```
cfontoptions:SetVariations(variations$)
```

FUNCTION

Sets the OpenType font variations for the font options object. Font variations are specified as a string with a format that is similar to the CSS font-variation-settings. The string contains a comma-separated list of axis assignments, which each assignment consists of a 4-character axis name and a value, separated by whitespace and optional equals sign.

Examples:

```
wght=200,wdth=140.5
wght 200 , wdth 140.5
```

INPUTS

`variations$`
: the new font variations

7.17 cfontoptions:Status

NAME

cfontoptions:Status – get font options status

SYNOPSIS

```
status = cfontoptions:Status()
```

FUNCTION

Checks whether an error has previously occurred for this font options object

This function returns `#CAIRO_STATUS_SUCCESS`, `#CAIRO_STATUS_NO_MEMORY`, or `#CAIRO_STATUS_NULL_POINTER`.

INPUTS

none

RESULTS

`status` a Cairo status code (see above)

8 Cairo glyphs

8.1 cglyphs:Free

NAME
cglyphs:Free – free glyphs array

SYNOPSIS
`cglyphs:Free()`

FUNCTION
Frees a glyphs array allocated by cairo.Glyphs().

INPUTS
none

8.2 cglyphs:Get

NAME
cglyphs:Get – get glyph index and offset

SYNOPSIS
`index, x, y = cglyphs:Get(n)`

FUNCTION
Returns information about the glyph at index `n` in the glyphs array. The `index` return value specifies the glyph index in the font. The exact interpretation of the glyph index depends on the font technology being used. `x` and `y` specify the offset in the X and Y direction between the origin used for drawing or measuring the string and the origin of this glyph.

INPUTS

`n`	which glyph to get

RESULTS

`index`	glyph ID
`x`	x offset of the glyph
`y`	y offset of the glyph

8.3 cglyphs:Set

NAME
cglyphs:Set – set glyph index and offset

SYNOPSIS
`cglyphs:Set(n, index, x, y)`

FUNCTION

Set information about the glyph at index `n` in the glyphs array. `index` specifies the glyph index in the font. The exact interpretation of the glyph index depends on the font technology being used. `x` and `y` specify the offset in the X and Y direction between the origin used for drawing or measuring the string and the origin of this glyph.

INPUTS

`n`	glyph to set
`index`	glyph ID
`x`	x offset of the glyph
`y`	y offset of the glyph

9 Cairo matrix

9.1 cmatrix:Get

NAME
cmatrix:Get – get matrix affine transformation

SYNOPSIS
```
xx, yx, xy, yy, x0, y0 = cmatrix:Get()
```

FUNCTION
Gets the affine transformation from the matrix and returns it.

INPUTS
none

RESULTS

`xx`	xx component of the affine transformation
`yx`	yx component of the affine transformation
`xy`	xy component of the affine transformation
`yy`	yy component of the affine transformation
`x0`	X translation component of the affine transformation
`y0`	Y translation component of the affine transformation

9.2 cmatrix:Init

NAME
cmatrix:Init – init matrix

SYNOPSIS
```
cmatrix:Init(xx, yx, xy, yy, x0, y0)
```

FUNCTION
Sets `matrix` to be the affine transformation given by `xx`, `yx`, `xy`, `yy`, `x0`, `y0`. The transformation is given by:

```
x_new = xx * x + xy * y + x0;
y_new = yx * x + yy * y + y0;
```

INPUTS

`xx`	xx component of the affine transformation
`yx`	yx component of the affine transformation
`xy`	xy component of the affine transformation
`yy`	yy component of the affine transformation
`x0`	X translation component of the affine transformation
`y0`	Y translation component of the affine transformation

9.3 cmatrix:InitIdentity

NAME
cmatrix:InitIdentity – init identity

SYNOPSIS
`cmatrix:InitIdentity()`

FUNCTION
Modifies the matrix to be an identity transformation.

INPUTS
none

9.4 cmatrix:InitRotate

NAME
cmatrix:InitRotate – init rotate

SYNOPSIS
`cmatrix:InitRotate(radians)`

FUNCTION
Initializes the matrix to a transformation that rotates by `radians`. The direction of rotation is defined such that positive angles rotate in the direction from the positive X axis toward the positive Y axis. With the default axis orientation of Cairo, positive angles rotate in a clockwise direction.

INPUTS

`radians`	angle of rotation in radians

9.5 cmatrix:InitScale

NAME
cmatrix:InitScale – init scale

SYNOPSIS
`cmatrix:InitScale(sx, sy)`

FUNCTION
Initializes the matrix to a transformation that scales by `sx` and `sy` in the X and Y dimensions, respectively.

INPUTS

`sx`	scale factor in the X direction
`sy`	scale factor in the Y direction

9.6 cmatrix:InitTranslate

NAME
cmatrix:InitTranslate – init translate

SYNOPSIS
```
cmatrix:InitTranslate(tx, ty)
```

FUNCTION
Initializes matrix to a transformation that translates by `tx` and `ty` in the X and Y dimensions, respectively.

INPUTS

`tx`	amount to translate in the X direction
`ty`	amount to translate in the Y direction

9.7 cmatrix:Invert

NAME
cmatrix:Invert – invert matrix

SYNOPSIS
```
status = cmatrix:Invert()
```

FUNCTION
Changes the matrix to be the inverse of its original value. Not all transformation matrices have inverses; if the matrix collapses points together (it is degenerate), then it has no inverse and this function will fail.

If the matrix has an inverse, modifies the matrix to be the inverse matrix and returns `#CAIRO_STATUS_SUCCESS`. Otherwise, returns `#CAIRO_STATUS_INVALID_MATRIX`.

INPUTS
none

RESULTS

`status`	status code indicating success or error (see above)

9.8 cmatrix:Multiply

NAME
cmatrix:Multiply – multiply matrix

SYNOPSIS
```
cmatrix:Multiply(a, b)
```

FUNCTION
Multiplies the affine transformations in `a` and `b` together and stores the result in the target matrix. The effect of the resulting transformation is to first apply the transformation in `a` to the coordinates and then apply the transformation in `b` to the coordinates.

It is allowable for the target matrix to be identical to either `a` or `b`.

INPUTS

`a`	a Cairo matrix
`b`	a Cairo matrix

9.9 cmatrix:Rotate

NAME
cmatrix:Rotate – rotate matrix

SYNOPSIS
`cmatrix:Rotate(radians)`

FUNCTION
Applies rotation by `radians` to the transformation in the matrix. The effect of the new transformation is to first rotate the coordinates by `radians`, then apply the original transformation to the coordinates.

The direction of rotation is defined such that positive angles rotate in the direction from the positive X axis toward the positive Y axis. With the default axis orientation of cairo, positive angles rotate in a clockwise direction.

INPUTS

`radians`	angle of rotation in radians

9.10 cmatrix:Scale

NAME
cmatrix:Scale – scale matrix

SYNOPSIS
`cmatrix:Scale(sx, sy)`

FUNCTION
Applies scaling by `sx`, `sy` to the transformation in the matrix. The effect of the new transformation is to first scale the coordinates by `sx` and `sy`, then apply the original transformation to the coordinates.

INPUTS

`sx`	scale factor in the X direction
`sy`	scale factor in the Y direction

9.11 cmatrix:TransformDistance

NAME
cmatrix:TransformDistance – transform distance vector

SYNOPSIS

```
tx, ty = cmatrix:TransformDistance(dx, dy)
```

FUNCTION

Transforms the distance vector (dx,dy) by the matrix. This is similar to cmatrix:TransformPoint() except that the translation components of the transformation are ignored. The calculation of the returned vector is as follows:

```
dx2 = dx1 * a + dy1 * c;
dy2 = dx1 * b + dy1 * d;
```

Affine transformations are position invariant, so the same vector always transforms to the same vector. If (x1,y1) transforms to (x2,y2) then (x1+dx1,y1+dy1) will transform to (x1+dx2,y1+dy2) for all values of x1 and x2.

INPUTS

dx	X component of a distance vector
dy	Y component of a distance vector

RESULTS

tx	transformed X component of a distance vector
ty	transformed Y component of a distance vector

9.12 cmatrix:TransformPoint

NAME

cmatrix:TransformPoint – transform point

SYNOPSIS

```
tx, ty = cmatrix:TransformPoint(x, y)
```

FUNCTION

Transforms the point (x, y) by the matrix.

INPUTS

x	X position
y	Y position

RESULTS

tx	transformed X position
ty	transformed Y position

9.13 cmatrix:Translate

NAME

cmatrix:Translate – translate matrix

SYNOPSIS

```
cmatrix:Translate(tx, ty)
```

FUNCTION

Applies a translation by `tx`, `ty` to the transformation in the matrix. The effect of the new transformation is to first translate the coordinates by `tx` and `ty`, then apply the original transformation to the coordinates.

INPUTS

`tx`	amount to translate in the X direction
`ty`	amount to translate in the Y direction

10 Cairo path

10.1 cpath:Free

NAME
cpath:Free – free path

SYNOPSIS
`cpath:Free()`

FUNCTION
Frees a path allocated by cairo.Path().

INPUTS
none

10.2 cpath:Get

NAME
cpath:Get – get path

SYNOPSIS
`t = cpath:Get()`

FUNCTION
Returns the individual items that make up the path. The individual path items are returned as a table containing a number of subtables each describing a single path item. See cairo.Path() for a description of the valid fields in the subtables.

INPUTS
none

RESULTS

`t`	table containing all path items

11 Cairo pattern

11.1 cpattern:AddColorStopRGB

NAME

cpattern:AddColorStopRGB – add RGB color stop

SYNOPSIS

```
cpattern:AddColorStopRGB(offset, red, green, blue)
```

FUNCTION

Adds an opaque color stop to a gradient pattern. The offset specifies the location along the gradient's control vector. For example, a linear gradient's control vector is from (x0,y0) to (x1,y1) while a radial gradient's control vector is from any point on the start circle to the corresponding point on the end circle.

The color is specified in the same way as in ccontext:SetSourceRGB().

If two (or more) stops are specified with identical offset values, they will be sorted according to the order in which the stops are added (stops added earlier will compare less than stops added later). This can be useful for reliably making sharp color transitions instead of the typical blend.

Note: If the pattern is not a gradient pattern, (e.g. a linear or radial pattern), then the pattern will be put into an error status with a status of `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`.

INPUTS

`offset`	an offset in the range [0.0 .. 1.0]
`red`	red component of color
`green`	green component of color
`blue`	blue component of color

11.2 cpattern:AddColorStopRGBA

NAME

cpattern:AddColorStopRGBA – add RGBA color stop

SYNOPSIS

```
cpattern:AddColorStopRGBA(offset, red, green, blue, alpha)
```

FUNCTION

Adds a translucent color stop to a gradient pattern. The offset specifies the location along the gradient's control vector. For example, a linear gradient's control vector is from (x0,y0) to (x1,y1) while a radial gradient's control vector is from any point on the start circle to the corresponding point on the end circle.

The color is specified in the same way as in ccontext:SetSourceRGBA().

If two (or more) stops are specified with identical offset values, they will be sorted according to the order in which the stops are added (stops added earlier will compare

less than stops added later). This can be useful for reliably making sharp color transitions instead of the typical blend.

Note: If the pattern is not a gradient pattern, (e.g. a linear or radial pattern), then the pattern will be put into an error status with a status of `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`.

INPUTS

`offset`	an offset in the range [0.0 .. 1.0]
`red`	red component of color
`green`	green component of color
`blue`	blue component of color
`alpha`	alpha component of color

11.3 cpattern:BeginPatch

NAME

cpattern:BeginPatch – begin patch

SYNOPSIS

```
cpattern:BeginPatch()
```

FUNCTION

Begin a patch in a mesh pattern.

After calling this function, the patch shape should be defined with cpattern:MoveTo(), cpattern:LineTo() and cpattern:CurveTo().

After defining the patch, cpattern:EndPatch() must be called before using the pattern as a source or mask.

Note: If the pattern is not a mesh pattern then it will be put into an error status with a status of `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`. If the pattern already has a current patch, it will be put into an error status with a status of `#CAIRO_STATUS_INVALID_MESH_CONSTRUCTION`.

INPUTS

none

11.4 cpattern:CurveTo

NAME

cpattern:CurveTo – add cubic Bezier spline

SYNOPSIS

```
cpattern:CurveTo(x1, y1, x2, y2, x3, y3)
```

FUNCTION

Adds a cubic Bezier spline to the current patch from the current point to position (`x3`, `y3`) in pattern-space coordinates, using (`x1`, `y1`) and (`x2`, `y2`) as the control points.

If the current patch has no current point before the call to cpattern:CurveTo(), this function will behave as if preceded by a call to cpattern:MoveTo() with `x1` and `y1` as parameters.

After this call the current point will be (`x3`, `y3`).

Note: If the pattern is not a mesh pattern then it will be put into an error status with a status of `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`. If the pattern has no current patch or the current patch already has 4 sides, it will be put into an error status with a status of `#CAIRO_STATUS_INVALID_MESH_CONSTRUCTION`.

INPUTS

`x1`	the X coordinate of the first control point
`y1`	the Y coordinate of the first control point
`x2`	the X coordinate of the second control point
`y2`	the Y coordinate of the second control point
`x3`	the X coordinate of the end of the curve
`y3`	the Y coordinate of the end of the curve

11.5 cpattern:EndPatch

NAME

cpattern:EndPatch – end patch

SYNOPSIS

```
cpattern:EndPatch()
```

FUNCTION

Indicates the end of the current patch in a mesh pattern.

If the current patch has less than 4 sides, it is closed with a straight line from the current point to the first point of the patch as if cpattern:LineTo() was used.

Note: If the pattern is not a mesh pattern then it will be put into an error status with a status of `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`. If the pattern has no current patch or the current patch already has 4 sides, it will be put into an error status with a status of `#CAIRO_STATUS_INVALID_MESH_CONSTRUCTION`.

INPUTS

none

11.6 cpattern:Free

NAME

cpattern:Free – free pattern

SYNOPSIS

```
cpattern:Free()
```

FUNCTION

Decreases the reference count on the pattern by one. If the result is zero, then the pattern and all associated resources are freed. See cpattern:Reference().

INPUTS

none

11.7 cpattern:GetColorStopCount

NAME

cpattern:GetColorStopCount – get color stop count

SYNOPSIS

```
status, count = cpattern:GetColorStopCount()
```

FUNCTION

Gets the number of color stops specified in the given gradient pattern.

This function returns `#CAIRO_STATUS_SUCCESS`, or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH` if the pattern is not a gradient pattern.

INPUTS

none

RESULTS

`status`	status code (see above)
`count`	number of color stops

11.8 cpattern:GetColorStopRGBA

NAME

cpattern:GetColorStopRGBA – get color stop RGBA

SYNOPSIS

```
status, offset, r, g, b, a = cpattern:GetColorStopRGBA(index)
```

FUNCTION

Gets the color and offset information at the given `index` for a gradient pattern. Values of `index` range from 0 to n-1 where n is the number returned by cpattern:GetColorStopCount().

Note that the color and alpha values are not premultiplied.

This function also returns `#CAIRO_STATUS_SUCCESS`, or `#CAIRO_STATUS_INVALID_INDEX` if `index` is not valid for the given pattern. If the pattern is not a gradient pattern, `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH` is returned.

INPUTS

`index`	index of the stop to return data for

RESULTS

`status`	`#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_INVALID_INDEX`

`offset`	offset of the stop
`r`	red component of color
`g`	green component of color
`b`	blue component of color
`a`	alpha component of color

11.9 cpattern:GetControlPoint

NAME

cpattern:GetControlPoint – get control point

SYNOPSIS

```
status, x, y = cpattern:GetControlPoint(patch_num, point_num)
```

FUNCTION

Gets the control point `point_num` of patch `patch_num` for a mesh pattern.

`patch_num` can range from 0 to n-1 where `n` is the number returned by cpattern:GetPatchCount().

Valid values for `point_num` are from 0 to 3 and identify the control points as explained in cairo.PatternMesh().

This function also returns `#CAIRO_STATUS_SUCCESS`, or `#CAIRO_STATUS_INVALID_INDEX` if `patch_num` or `point_num` is not valid for the pattern. If the pattern is not a mesh pattern, `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH` is returned.

INPUTS

`patch_num`
: the patch number to return data for

`point_num`
: the control point number to return data for

RESULTS

`status`	`#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_INVALID_INDEX`
`x`	x coordinate of the control point
`y`	y coordinate of the control point

11.10 cpattern:GetCornerColorRGBA

NAME

cpattern:GetCornerColorRGBA – get corner color RGBA

SYNOPSIS

```
status, r, g, b, a = cpattern:GetCornerColorRGBA(patch_num, corner_num)
```

FUNCTION

Gets the color information in corner `corner_num` of patch `patch_num` for a mesh pattern.

`patch_num` can range from 0 to n-1 where `n` is the number returned by cpattern:GetPatchCount().

Valid values for `corner_num` are from 0 to 3 and identify the corners as explained in cairo.PatternMesh().

Note that the color and alpha values are not premultiplied.

This function also returns `#CAIRO_STATUS_SUCCESS`, or `#CAIRO_STATUS_INVALID_INDEX` if `patch_num` or `corner_num` is not valid for the pattern. If the pattern is not a mesh pattern, `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH` is returned.

INPUTS

`patch_num`
: the patch number to return data for

`corner_num`
: the corner number to return data for

RESULTS

`status`	`#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_INVALID_INDEX`
`r`	red component of color
`g`	green component of color
`b`	blue component of color
`a`	alpha component of color

11.11 cpattern:GetExtend

NAME

cpattern:GetExtend – get extend mode

SYNOPSIS

```
mode = cpattern:GetExtend()
```

FUNCTION

Gets the current extend mode for a pattern. See cpattern:SetExtend() for details on the semantics of each extend strategy.

INPUTS

none

RESULTS

`mode`	the current extend strategy used for drawing the pattern

11.12 cpattern:GetFilter

NAME
cpattern:GetFilter – get filter

SYNOPSIS
```
filter = cpattern:GetFilter()
```

FUNCTION
Gets the current filter for a pattern. See cpattern:SetFilter() for details on each filter.

INPUTS
none

RESULTS

`filter`	the current filter used for resizing the pattern

11.13 cpattern:GetLinearPoints

NAME
cpattern:GetLinearPoints – get gradient endpoints

SYNOPSIS
```
status, x0, y0, x1, y1 = cpattern:GetLinearPoints()
```

FUNCTION
Gets the gradient endpoints for a linear gradient.

This function also returns `#CAIRO_STATUS_SUCCESS`, or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH` if the pattern is not a linear gradient pattern.

INPUTS
none

RESULTS

`status`	`#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`
`x0`	x coordinate of the first point
`y0`	y coordinate of the first point
`x1`	x coordinate of the second point
`y1`	y coordinate of the second point

11.14 cpattern:GetMatrix

NAME
cpattern:GetMatrix – get pattern's transformation matrix

SYNOPSIS
```
m = cpattern:GetMatrix(matrix)
```

FUNCTION
Returns the pattern's transformation matrix.

INPUTS
none

RESULTS

`m`	current transformation matrix

11.15 cpattern:GetPatchCount

NAME
cpattern:GetPatchCount – get patch count

SYNOPSIS
```
status, count = cpattern:GetPatchCount()
```

FUNCTION
Gets the number of patches specified in the given mesh pattern.

The number only includes patches which have been finished by calling cpattern:EndPatch(). For example it will be 0 during the definition of the first patch.

This function also returns `#CAIRO_STATUS_SUCCESS`, or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH` if the pattern is not a mesh pattern.

INPUTS
none

RESULTS

`status`	`#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`
`count`	number of patches

11.16 cpattern:GetRadialCircles

NAME
cpattern:GetRadialCircles – get radial circles

SYNOPSIS
```
status, x0, y0, r0, x1, y1, r1 = cpattern:GetRadialCircles()
```

FUNCTION
Gets the gradient endpoint circles for a radial gradient, each specified as a center coordinate and a radius.

This function also returns `#CAIRO_STATUS_SUCCESS`, or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH` if the pattern is not a radial gradient pattern.

INPUTS
none

RESULTS

`x`	`#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`
`x0`	x coordinate of the center of the first circle
`y0`	y coordinate of the center of the first circle
`r0`	radius of the first circle
`x1`	x coordinate of the center of the second circle
`y1`	y coordinate of the center of the second circle
`r1`	radius of the second circle

11.17 cpattern:GetReferenceCount

NAME

cpattern:GetReferenceCount – get reference count

SYNOPSIS

```
count = cpattern:GetReferenceCount()
```

FUNCTION

Returns the current reference count of the pattern.

This function returns the current reference count of the pattern. If the object is a `Nil` object, 0 will be returned.

INPUTS

none

RESULTS

`count`	the current reference count of the pattern

11.18 cpattern:GetRGBA

NAME

cpattern:GetRGBA – get RGBA

SYNOPSIS

```
status, red, green, blue, alpha = cpattern:GetRGBA()
```

FUNCTION

Gets the solid color for a solid color pattern.

Note that the color and alpha values are not premultiplied.

This function also returns `#CAIRO_STATUS_SUCCESS`, or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH` if the pattern is not a solid color pattern.

INPUTS

none

RESULTS

`status`	`#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`
`red`	red component of color
`green`	green component of color
`blue`	blue component of color
`alpha`	alpha component of color

11.19 cpattern:GetSurface

NAME

cpattern:GetSurface – get pattern surface

SYNOPSIS

```
status, surface = cpattern:GetSurface()
```

FUNCTION

Gets the surface of a surface pattern. The reference returned is owned by the pattern; the caller should call csurface:Reference() if the surface is to be retained.

This function also returns `#CAIRO_STATUS_SUCCESS`, or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH` if the pattern is not a surface pattern.

INPUTS

none

RESULTS

`status`	`#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`
`surface`	surface of pattern

11.20 cpattern:GetType

NAME

cpattern:GetType – get type

SYNOPSIS

```
type = cpattern:GetType()
```

FUNCTION

Get the pattern's type. This will be one of the following types:

`#CAIRO_PATTERN_TYPE_SOLID`
: The pattern is a solid (uniform) color. It may be opaque or translucent.

`#CAIRO_PATTERN_TYPE_SURFACE`
: The pattern is a based on a surface (an image).

`#CAIRO_PATTERN_TYPE_LINEAR`
: The pattern is a linear gradient.

`#CAIRO_PATTERN_TYPE_RADIAL`
: The pattern is a radial gradient.

`#CAIRO_PATTERN_TYPE_MESH`
: The pattern is a mesh.

`#CAIRO_PATTERN_TYPE_RASTER_SOURCE`
: The pattern is a user pattern providing raster data.

INPUTS

none

RESULTS

`type` the type of the pattern

11.21 cpattern:IsNull

NAME

cpattern:IsNull – check if pattern is invalid

SYNOPSIS

```
bool = cpattern:IsNull()
```

FUNCTION

Returns `True` if the pattern is `NULL`, i.e. invalid. If functions that allocate patterns fail, they might not throw an error but simply set the pattern to `NULL`. You can use this function to check if pattern allocation has failed in which case the pattern will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the pattern is `NULL`, otherwise `False`

11.22 cpattern:LineTo

NAME

cpattern:LineTo – add line to patch

SYNOPSIS

```
cpattern:LineTo(x, y)
```

FUNCTION

Adds a line to the current patch from the current point to position (`x`, `y`) in pattern-space coordinates.

If there is no current point before the call to cpattern:LineTo() this function will behave as cpattern:MoveTo() with the arguments `x` and `y`.

After this call the current point will be (x, y).

Note: If the pattern is not a mesh pattern then it will be put into an error status with a status of #CAIRO_STATUS_PATTERN_TYPE_MISMATCH. If the pattern has no current patch or the current patch already has 4 sides, it will be put into an error status with a status of #CAIRO_STATUS_INVALID_MESH_CONSTRUCTION.

INPUTS

x	the X coordinate of the end of the new line
y	the Y coordinate of the end of the new line

11.23 cpattern:MoveTo

NAME

cpattern:MoveTo – move to point

SYNOPSIS

```
cpattern:MoveTo(x, y)
```

FUNCTION

Define the first point of the current patch in a mesh pattern.

After this call the current point will be (x, y).

Note: If the pattern is not a mesh pattern then it will be put into an error status with a status of #CAIRO_STATUS_PATTERN_TYPE_MISMATCH. If the pattern has no current patch or the current patch already has 4 sides, it will be put into an error status with a status of #CAIRO_STATUS_INVALID_MESH_CONSTRUCTION.

INPUTS

x	the X coordinate of the new position
y	the Y coordinate of the new position

11.24 cpattern:Reference

NAME

cpattern:Reference – increase reference count

SYNOPSIS

```
cpattern:Reference()
```

FUNCTION

Increases the reference count on the pattern by one. This prevents the pattern from being destroyed until a matching call to cpattern:Free() is made.

Use cpattern:GetReferenceCount() to get the number of references to a Cairo pattern.

INPUTS

none

11.25 cpattern:SetControlPoint

NAME

cpattern:SetControlPoint – set control point

SYNOPSIS

```
cpattern:SetControlPoint(point_num, x, y)
```

FUNCTION

Set an internal control point of the current patch.

Valid values for `point_num` are from 0 to 3 and identify the control points as explained in cairo.PatternMesh().

Note: If the pattern is not a mesh pattern then it will be put into an error status with a status of `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`. If `point_num` is not valid, the pattern will be put into an error status with a status of `#CAIRO_STATUS_INVALID_INDEX`. If the pattern has no current patch, it will be put into an error status with a status of `#CAIRO_STATUS_INVALID_MESH_CONSTRUCTION`.

INPUTS

`point_num`
: the control point to set the position for

`x`
: the X coordinate of the control point

`y`
: the Y coordinate of the control point

11.26 cpattern:SetCornerColorRGB

NAME

cpattern:SetCornerColorRGB – set corner color RGB

SYNOPSIS

```
cpattern:SetCornerColorRGB(corner_num, red, green, blue)
```

FUNCTION

Sets the color of a corner of the current patch in a mesh pattern.

The color is specified in the same way as in ccontext:SetSourceRGB().

Valid values for `corner_num` are from 0 to 3 and identify the corners as explained in cairo.PatternMesh().

Note: If the pattern is not a mesh pattern then it will be put into an error status with a status of `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`. If `corner_num` is not valid, the pattern will be put into an error status with a status of `#CAIRO_STATUS_INVALID_INDEX`. If the pattern has no current patch, it will be put into an error status with a status of `#CAIRO_STATUS_INVALID_MESH_CONSTRUCTION`.

INPUTS

`corner_num`
: the corner to set the color for

`red`
: red component of color

`green` green component of color

`blue` blue component of color

11.27 cpattern:SetCornerColorRGBA

NAME

cpattern:SetCornerColorRGBA – set corner color RGBA

SYNOPSIS

```
cpattern:SetCornerColorRGBA(corner_num, red, green, blue, alpha)
```

FUNCTION

Sets the color of a corner of the current patch in a mesh pattern.

The color is specified in the same way as in ccontext:SetSourceRGBA().

Valid values for `corner_num` are from 0 to 3 and identify the corners as explained in cairo.PatternMesh().

Note: If the pattern is not a mesh pattern then it will be put into an error status with a status of `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`. If `corner_num` is not valid, the pattern will be put into an error status with a status of `#CAIRO_STATUS_INVALID_INDEX`. If the pattern has no current patch, it will be put into an error status with a status of `#CAIRO_STATUS_INVALID_MESH_CONSTRUCTION`.

INPUTS

`corner_num`
the corner to set the color for

`red` red component of color

`green` green component of color

`blue` blue component of color

`alpha` alpha component of color

11.28 cpattern:SetExtend

NAME

cpattern:SetExtend – set extend mode

SYNOPSIS

```
cpattern:SetExtend(extend)
```

FUNCTION

Sets the mode to be used for drawing outside the area of a pattern. The `extend` parameter can be set to one of the following constants:

`#CAIRO_EXTEND_NONE`
Pixels outside of the source pattern are fully transparent.

`#CAIRO_EXTEND_REPEAT`
: The pattern is tiled by repeating.

`#CAIRO_EXTEND_REFLECT`
: The pattern is tiled by reflecting at the edges.

`#CAIRO_EXTEND_PAD`
: Pixels outside of the pattern copy the closest pixel from the source.

The default extend mode is `#CAIRO_EXTEND_NONE` for surface patterns and `#CAIRO_EXTEND_PAD` for gradient patterns.

INPUTS

`extend` extend mode describing how the area outside of the pattern will be drawn (see above)

11.29 cpattern:SetFilter

NAME

cpattern:SetFilter – set filter

SYNOPSIS

```
cpattern:SetFilter(filter)
```

FUNCTION

Sets the filter to be used for resizing when using this pattern. The `filter` parameter can be set to one of the following constants:

`#CAIRO_FILTER_FAST`
: A high-performance filter, with quality similar to `#CAIRO_FILTER_NEAREST`.

`#CAIRO_FILTER_GOOD`
: A reasonable-performance filter, with quality similar to `#CAIRO_FILTER_BILINEAR`.

`#CAIRO_FILTER_BEST`
: The highest-quality available, performance may not be suitable for interactive use.

`#CAIRO_FILTER_NEAREST`
: Nearest-neighbor filtering.

`#CAIRO_FILTER_BILINEAR`
: Linear interpolation in two dimensions.

`#CAIRO_FILTER_GAUSSIAN`
: This filter value is currently unimplemented, and should not be used in current code.

Note that you might want to control filtering even when you do not have an explicit Cairo pattern object, for example when using ccontext:SetSourceSurface(). In these cases, it is convenient to use ccontext:GetSource() to get access to the pattern that cairo creates implicitly. For example:

```
ctx:SetSourceSurface(image, x, y)
```

```
pat = ctx:GetSource()
pat:SetFilter(#CAIRO_FILTER_NEAREST)
```

INPUTS

`filter` filter to use for resizing the pattern

11.30 cpattern:SetMatrix

NAME

cpattern:SetMatrix – set matrix

SYNOPSIS

`cpattern:SetMatrix(matrix)`

FUNCTION

Sets the pattern's transformation matrix to `matrix`. This matrix is a transformation from user space to pattern space.

When a pattern is first created it always has the identity matrix for its transformation matrix, which means that pattern space is initially identical to user space.

Important: Please note that the direction of this transformation matrix is from user space to pattern space. This means that if you imagine the flow from a pattern to user space (and on to device space), then coordinates in that flow will be transformed by the inverse of the pattern matrix.

For example, if you want to make a pattern appear twice as large as it does by default the correct code to use is:

```
m = cairo.Matrix()
m:InitScale(0.5, 0.5)
pat:SetMatrix(m)
```

Meanwhile, using values of 2.0 rather than 0.5 in the code above would cause the pattern to appear at half of its default size.

Also, please note the discussion of the user-space locking semantics of ccontext:SetSource().

INPUTS

`matrix` a Cairo matrix object

11.31 cpattern:Status

NAME

cpattern:Status – get pattern status

SYNOPSIS

`cpattern:Status()`

FUNCTION

Checks whether an error has previously occurred for this pattern.

This function returns `#CAIRO_STATUS_SUCCESS`, `#CAIRO_STATUS_NO_MEMORY`, `#CAIRO_STATUS_INVALID_MATRIX`, `#CAIRO_STATUS_PATTERN_TYPE_MISMATCH`, or `#CAIRO_STATUS_INVALID_MESH_CONSTRUCTION`.

INPUTS

none

RESULTS

`status` a Cairo status value (see above)

12 Cairo region

12.1 cregion:ContainsPoint

NAME
cregion:ContainsPoint – check if point is in region

SYNOPSIS
```
ok = cregion:ContainsPoint(x, y)
```

FUNCTION
Checks whether (`x`, `y`) is contained in the region.

INPUTS

`x`	the x coordinate of a point
`y`	the y coordinate of a point

RESULTS

`ok`	`True` if (`x`, `y`) is contained in the region, `False` if it is not

12.2 cregion:ContainsRectangle

NAME
cregion:ContainsRectangle – check if rectangle is in region

SYNOPSIS
```
overlap = cregion:ContainsRectangle(rectangle)
```

FUNCTION
Checks whether `rectangle` is inside, outside or partially contained in the region. The `rectangle` parameter must be a table that has the fields `x`, `y`, `width`, and `height` initialized.

This function returns `#CAIRO_REGION_OVERLAP_IN` if `rectangle` is entirely inside the region, `#CAIRO_REGION_OVERLAP_OUT` if `rectangle` is entirely outside the region, or `#CAIRO_REGION_OVERLAP_PART` if `rectangle` is partially inside and partially outside the region.

INPUTS

`rectangle`	a table describing a rectangle (see above)

RESULTS

`overlap`	an overlap constant (see above)

12.3 cregion:Copy

NAME
cregion:Copy – copy region

SYNOPSIS
```
reg = cregion:Copy()
```

FUNCTION
Allocates a new region object copying the area from the original.

This function returns a newly allocated Cairo region. Free with cregion:Free(). This function always returns a valid handle; if memory cannot be allocated, then a special error object is returned where all operations on the object do nothing. You can check for this with cregion:Status().

INPUTS
none

RESULTS

`reg`	a newly allocated Cairo region

12.4 cregion:Equal

NAME
cregion:Equal – check region equality

SYNOPSIS
```
ok = cregion:Equal(region_b)
```

FUNCTION
Compares whether `region_b` is equivalent to the region.

This function returns `True` if both regions contained the same coverage, `False` if it is not or any region is in an error status.

INPUTS

`region_b`	a Cairo region

RESULTS

`ok`	`True` if both regions contained the same coverage, `False` otherwise

12.5 cregion:Free

NAME
cregion:Free – free region

SYNOPSIS
```
cregion:Free()
```

FUNCTION
Destroys a Cairo region object created with cairo.Region() or cregion:Copy().

INPUTS
none

12.6 cregion:GetExtents

NAME
cregion:GetExtents – get region extents

SYNOPSIS
`rc = cregion:GetExtents()`

FUNCTION
Gets the bounding rectangle of the region. This will return a table that has the `x`, `y`, `width`, and `height` fields initialized.

INPUTS
none

RESULTS

`rc`	bounding rectangle of the region

12.7 cregion:GetRectangle

NAME
cregion:GetRectangle – get rectangle from region

SYNOPSIS
`rc = cregion:GetRectangle(nth)`

FUNCTION
Returns the `nth` rectangle from the region. This will return a table that has the `x`, `y`, `width`, and `height` fields initialized.

INPUTS

`nth`	a number indicating which rectangle should be returned

RESULTS

`rc`	rectangle at the specified index

12.8 cregion:Intersect

NAME
cregion:Intersect – intersect regions

SYNOPSIS
`status = cregion:Intersect(other)`

FUNCTION
Computes the intersection of the region with `other`.

INPUTS

`other` another Cairo region

RESULTS

`ok` `#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_NO_MEMORY`

12.9 cregion:IntersectRectangle

NAME

cregion:IntersectRectangle – intersect rectangle

SYNOPSIS

```
status = cregion:IntersectRectangle(rectangle)
```

FUNCTION

Computes the intersection of the region with `rectangle`. The `rectangle` parameter must be a table that has the fields `x`, `y`, `width`, and `height` initialized.

INPUTS

`rectangle`
a table containing a rectangle (see above)

RESULTS

`status` `#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_NO_MEMORY`

12.10 cregion:IsEmpty

NAME

cregion:IsEmpty – check if region is empty

SYNOPSIS

```
ok = cregion:IsEmpty()
```

FUNCTION

Checks whether the region is empty.

INPUTS

none

RESULTS

`ok` `True` if the region is empty, `False` if it isn't

12.11 cregion:IsNull

NAME

cregion:IsNull – check if region is invalid

SYNOPSIS

```
bool = cregion:IsNull()
```

FUNCTION

Returns `True` if the region is `NULL`, i.e. invalid. If functions that allocate regions fail, they might not throw an error but simply set the region to `NULL`. You can use this function to check if region allocation has failed in which case the region will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool`	`True` if the region is `NULL`, otherwise `False`

12.12 cregion:NumRectangles

NAME

cregion:NumRectangles – num rectangles

SYNOPSIS

```
count = cregion:NumRectangles()
```

FUNCTION

Returns the number of rectangles contained in the region.

INPUTS

none

RESULTS

`count`	the number of rectangles contained in the region

12.13 cregion:Reference

NAME

cregion:Reference – increase reference count

SYNOPSIS

```
cregion:Reference()
```

FUNCTION

Increases the reference count on the region by one. This prevents the region from being destroyed until a matching call to cregion:Free() is made.

INPUTS

none

12.14 cregion:Status

NAME
cregion:Status – get region status

SYNOPSIS
`status = cregion:Status()`

FUNCTION
Checks whether an error has previous occurred for this region object.

INPUTS
none

RESULTS
`status` a Cairo status code

12.15 cregion:Subtract

NAME
cregion:Subtract – subtract region

SYNOPSIS
`status = cregion:Subtract(other)`

FUNCTION
Subtracts the region specified by `other` from the region.

INPUTS
`other` another Cairo region

RESULTS
`status` `#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_NO_MEMORY`

12.16 cregion:SubtractRectangle

NAME
cregion:SubtractRectangle – subtract rectangle

SYNOPSIS
`status = cregion:SubtractRectangle(rectangle)`

FUNCTION
Subtracts `rectangle` from the region. The `rectangle` parameter must be a table that has the fields `x`, `y`, `width`, and `height` initialized.

INPUTS
`rectangle`
a table containing a rectangle (see above)

RESULTS
`x` `#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_NO_MEMORY`

12.17 cregion:Translate

NAME
cregion:Translate – translate region

SYNOPSIS
```
cregion:Translate(dx, dy)
```

FUNCTION
Translates the region by (`dx`, `dy`).

INPUTS

`dx`	Amount to translate in the x direction
`dy`	Amount to translate in the y direction

12.18 cregion:Union

NAME
cregion:Union – compute region union

SYNOPSIS
```
status = cregion:Union(other)
```

FUNCTION
Computes the union of the current region and the region specified by `other`.

INPUTS

`other`	another Cairo region

RESULTS

`status`	`#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_NO_MEMORY`

12.19 cregion:UnionRectangle

NAME
cregion:UnionRectangle – compute region union with rectangle

SYNOPSIS
```
status = cregion:UnionRectangle(rectangle)
```

FUNCTION
Computes the union of the current region and the rectangle specified by `rectangle`. The `rectangle` parameter must be a table that has the fields `x`, `y`, `width`, and `height` initialized.

INPUTS

`rectangle`	a table containing a rectangle (see above)

RESULTS

`status`	`#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_NO_MEMORY`

12.20 cregion:Xor

NAME

cregion:Xor – compute exclusive difference of region

SYNOPSIS

```
status = cregion:Xor(other)
```

FUNCTION

Computes the exclusive difference of the current region with the region specified by `other`. That is, the current region will be set to contain all areas that are either in the current region or in `other`, but not in both.

INPUTS

`other` another Cairo region

RESULTS

`status` `#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_NO_MEMORY`

12.21 cregion:XorRectangle

NAME

cregion:XorRectangle – compute exclusive difference of region with rectangle

SYNOPSIS

```
status = cregion:XorRectangle(rectangle)
```

FUNCTION

Computes the exclusive difference of the current region with `rectangle`. That is, the current region will be set to contain all areas that are either in the current region or in `rectangle`, but not in both. The `rectangle` parameter must be a table that has the fields `x`, `y`, `width`, and `height` initialized.

INPUTS

`rectangle`
a table containing a rectangle (see above)

RESULTS

`status` `#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_NO_MEMORY`

13 Cairo scaled font

13.1 cscaledfont:Extents

NAME
cscaledfont:Extents – get font extents

SYNOPSIS
```
t = cscaledfont:Extents(extents)
```

FUNCTION
Gets the metrics for a Cairo scaled font. This returns a table containing the font extents. See ccontext:FontExtents() for an explanation of the individual table members.

INPUTS

`t`	table containing the font extents (see above)

13.2 cscaledfont:Free

NAME
cscaledfont:Free – free scaled font

SYNOPSIS
```
cscaledfont:Free()
```

FUNCTION
Decreases the reference count on the font by one. If the result is zero, then the font and all associated resources are freed. See cscaledfont:Reference().

INPUTS
none

13.3 cscaledfont:GetCTM

NAME
cscaledfont:GetCTM – get current transformation matrix

SYNOPSIS
```
m = cscaledfont:GetCTM()
```

FUNCTION
Returns the CTM with which the scaled font was created. Note that the translation offsets (x0, y0) of the CTM are ignored by cairo.ScaledFont(). So, the matrix this function returns always has 0,0 as x0,y0.

INPUTS

`m`	current transformation matrix

13.4 cscaledfont:GetFontFace

NAME
cscaledfont:GetFontFace – get font face

SYNOPSIS
```
font = cscaledfont:GetFontFace()
```

FUNCTION
Gets the font face that this scaled font uses. This might be the font face passed to cairo.ScaledFont(), but this does not hold true for all possible cases.

This function returns the Cairo font face with which the scaled font was created. This object is owned by cairo. To keep a reference to it, you must call cscaledfont:Reference().

INPUTS
none

RESULTS

`font`	the Cairo font face with which the scaled font was created

13.5 cscaledfont:GetFontMatrix

NAME
cscaledfont:GetFontMatrix – get font matrix

SYNOPSIS
```
m = cscaledfont:GetFontMatrix()
```

FUNCTION
Returns the font matrix with which the scaled font was created.

INPUTS

`m`	a Cairo matrix

13.6 cscaledfont:GetFontOptions

NAME
cscaledfont:GetFontOptions – get font options

SYNOPSIS
```
opt = cscaledfont:GetFontOptions(options)
```

FUNCTION
Returns the font options with which the scaled font was created.

INPUTS

`opt`	a Cairo font options object

13.7 cscaledfont:GetReferenceCount

NAME
cscaledfont:GetReferenceCount – get reference count

SYNOPSIS
```
count = cscaledfont:GetReferenceCount()
```

FUNCTION
Returns the current reference count of the scaled font.

INPUTS
none

RESULTS

`count`	the current reference count of the scaled font

13.8 cscaledfont:GetScaleMatrix

NAME
cscaledfont:GetScaleMatrix – get scale matrix

SYNOPSIS
```
m = cscaledfont:GetScaleMatrix()
```

FUNCTION
Returns the scale matrix of the scaled font. The scale matrix is product of the font matrix and the ctm associated with the scaled font, and hence is the matrix mapping from font space to device space.

INPUTS

`m`	scale matrix of the font

13.9 cscaledfont:GetType

NAME
cscaledfont:GetType – get font type

SYNOPSIS
```
type = cscaledfont:GetType()
```

FUNCTION
This function returns the type of the backend used to create a scaled font. See cfontface:GetType() for a list of possible font types. Note that this function never returns `#CAIRO_FONT_TYPE_TOY`.

INPUTS
none

RESULTS

`type`	the type of the scaled font

13.10 cscaledfont:GlyphExtents

NAME

cscaledfont:GlyphExtents – get glyph extents

SYNOPSIS

```
t = cscaledfont:GlyphExtents(glyphs[, offset, num_glyphs])
```

FUNCTION

Gets the extents for a glyphs array. The extents describe a user-space rectangle that encloses the "inked" portion of the glyphs as they would be drawn by ccontext:ShowGlyphs() if the cairo graphics state were set to the same font face, font matrix, ctm, and font options as the scaled font. Additionally, the `XAdvance` and `YAdvance` values indicate the amount by which the current point would be advanced by ccontext:ShowGlyphs().

Note that whitespace glyphs do not contribute to the size of the rectangle as returned in the `Width` and `Height` members of the extents.

This function returns a table containing the glyph extents. See ccontext:TextExtents() for a description of all table members.

INPUTS

`glyphs` a glyphs array containinig glyph IDs with X and Y offsets

`offset` optional: offset into the array that specifies the starting glyph (defaults to 0 which means first glyph)

`num_glyphs` optional: number of glyphs to show (defaults to -1 which means all glyphs)

RESULTS

`t` table containing the glyph extents (see above)

13.11 cscaledfont:IsNull

NAME

cscaledfont:IsNull – check if scaled font is invalid

SYNOPSIS

```
bool = cscaledfont:IsNull()
```

FUNCTION

Returns `True` if the scaled font is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the object will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the object is `NULL`, otherwise `False`

13.12 cscaledfont:Reference

NAME

cscaledfont:Reference – increase reference count

SYNOPSIS

```
cscaledfont:Reference()
```

FUNCTION

Increases the reference count on the scaled font by one. This prevents the scaled font from being destroyed until a matching call to cscaledfont:Free() is made.

Use cscaledfont:GetReferenceCount() to get the number of references to a Cairo scaled font.

INPUTS

none

13.13 cscaledfont:Status

NAME

cscaledfont:Status – get scaled font status

SYNOPSIS

```
status = cscaledfont:Status()
```

FUNCTION

Checks whether an error has previously occurred for this scaled font.

This function returns `#CAIRO_STATUS_SUCCESS` or another error such as `#CAIRO_STATUS_NO_MEMORY`.

INPUTS

none

RESULTS

`status` a Cairo status code (see above)

13.14 cscaledfont:TextExtents

NAME

cscaledfont:TextExtents – get text extents

SYNOPSIS

```
t = cscaledfont:TextExtents(s$)
```

FUNCTION

Gets the extents for a string of text. The extents describe a user-space rectangle that encloses the "inked" portion of the text drawn at the origin (0,0) as it would be drawn by ccontext:ShowText() if the Cairographics state were set to the same font face, font matrix, ctm, and font options as the scaled font. Additionally, the `XAdvance` and `YAdvance` values indicate the amount by which the current point would be advanced by ccontext:ShowText().

Note that whitespace characters do not directly contribute to the size of the rectangle as contained in the `Width` and `Height` fields of the extents. They do contribute indirectly by changing the position of non-whitespace characters. In particular, trailing whitespace characters are likely to not affect the size of the rectangle, though they will affect the `XAdvance` and `YAdvance` values.

This function returns a table containing the text extents. See ccontext:TextExtents() for a description of all table members.

INPUTS

`s$`	a string of text

RESULTS

`t`	table containing the text extents (see above)

14 Cairo surface

14.1 csurface:AddOutline

NAME
csurface:AddOutline – add outline

SYNOPSIS
`id = csurface:AddOutline(parent_id, name$, link$, flags)`

FUNCTION
For PDF surfaces, this adds an item to the document outline hierarchy with the name `name$` that links to the location specified by `link$`. Link attributes have the same keys and values as the link tag, excluding the `rect` attribute. The item will be a child of the item with id `parent_id`. Use `#CAIRO_PDF_OUTLINE_ROOT` as the parent id of top level items. The `flags` parameter can be set to a combination of the following flags:

`#CAIRO_PDF_OUTLINE_FLAG_OPEN`
The outline item defaults to open in the PDF viewer.

`#CAIRO_PDF_OUTLINE_FLAG_BOLD`
The outline item is displayed by the viewer in bold text.

`#CAIRO_PDF_OUTLINE_FLAG_ITALIC`
The outline item is displayed by the viewer in italic text.

This function returns the id for the added item.

INPUTS

`parent_id` the id of the parent item or `#CAIRO_PDF_OUTLINE_ROOT` if this is a top level item

`name$` the name of the outline

`link$` the link attributes specifying where this outline links to

`flags` outline item flags (see above)

RESULTS

`id` id for the added item

14.2 csurface:CopyPage

NAME
csurface:CopyPage – copy page

SYNOPSIS
`csurface:CopyPage()`

FUNCTION

Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page. Use csurface:ShowPage() if you want to get an empty page after the emission.

There is a convenience function for this that takes a Cairo context, namely ccontext:CopyPage().

INPUTS

none

14.3 csurface:CreateForRectangle

NAME

csurface:CreateForRectangle – create surface from rectangle

SYNOPSIS

```
handle = csurface:CreateForRectangle(x, y, width, height)
```

FUNCTION

Create a new surface that is a rectangle within the target surface. All operations drawn to this surface are then clipped and translated onto the target surface. Nothing drawn via this sub-surface outside of its bounds is drawn onto the target surface, making this a useful method for passing constrained child surfaces to library routines that draw directly onto the parent surface, i.e. with no further backend allocations, double buffering or copies.

The semantics of subsurfaces have not been finalized yet unless the rectangle is in full device units, is contained within the extents of the target surface, and the target or subsurface's device transforms are not changed.

This function returns a handle to the newly allocated surface. The caller owns the surface and should call csurface:Free() when done with it.

This function always returns a valid handle, but it will return a handle to a "nil" surface if the source surface is already in an error state or any other error occurs.

INPUTS

`x`	the x-origin of the sub-surface from the top-left of the target surface (in device-space units)
`y`	the y-origin of the sub-surface from the top-left of the target surface (in device-space units)
`width`	width of the sub-surface (in device-space units)
`height`	height of the sub-surface (in device-space units)

RESULTS

`handle`	handle to a new surface

14.4 csurface:CreateSimilar

NAME

csurface:CreateSimilar – create similar surface

SYNOPSIS

```
handle = csurface:CreateSimilar(content, width, height)
```

FUNCTION

Create a new surface that is as compatible as possible with an existing surface. For example the new surface will have the same device scale, fallback resolution and font options as the source surface. Generally, the new surface will also use the same backend as the source surface, unless that is not possible for some reason. The type of the returned surface may be examined with csurface:GetType().

The `content` parameter can be set to one of the following constants:

`#CAIRO_CONTENT_COLOR`
: The surface will hold color content only.

`#CAIRO_CONTENT_ALPHA`
: The surface will hold alpha content only.

`#CAIRO_CONTENT_COLOR_ALPHA`
: The surface will hold color and alpha content.

Initially the surface contents are all 0 (transparent if contents have transparency, black otherwise.)

Use csurface:CreateSimilarImage() if you need an image surface which can be painted quickly to the target surface.

This function returns a handle to the newly allocated surface. The caller owns the surface and should call csurface:Free() when done with it.

This function always returns a valid handle, but it will return a handle to a "nil" surface if the source surface is already in an error state or any other error occurs.

INPUTS

`content`	the content for the new surface (see above)
`width`	width of the new surface, (in device-space units)
`height`	height of the new surface (in device-space units)

RESULTS

`handle`	handle to a new surface

14.5 csurface:CreateSimilarImage

NAME

csurface:CreateSimilarImage – create similar image surface

SYNOPSIS

```
handle = csurface:CreateSimilarImage(format, width, height)
```

FUNCTION

Create a new image surface that is as compatible as possible for uploading to and the use in conjunction with an existing surface. However, this surface can still be used like any normal image surface. Unlike csurface:CreateSimilar() the new image surface won't inherit the device scale from the source surface. The `format` parameter must be set to a pixel format constant. See Section 4.5 [cairo.ImageSurface], page 16, for details.

Initially the surface contents are all 0 (transparent if contents have transparency, black otherwise.)

Use csurface:CreateSimilar() if you don't need an image surface.

This function returns a handle to the newly allocated image surface. The caller owns the surface and should call csurface:Free() when done with it.

This function always returns a valid handle, but it will return a handle to a "nil" surface if `other` is already in an error state or any other error occurs.

INPUTS

`format`	the format for the new surface
`width`	width of the new surface, (in pixels)
`height`	height of the new surface (in pixels)

RESULTS

`handle`	handle to a new surface

14.6 csurface:Finish

NAME

csurface:Finish – finish surface

SYNOPSIS

```
csurface:Finish()
```

FUNCTION

This function finishes the surface and drops all references to external resources. For example, for the Xlib backend it means that Cairo will no longer access the drawable, which can be freed. After calling csurface:Finish() the only valid operations on a surface are checking status, getting and setting user, referencing and destroying, and flushing and finishing it. Further drawing to the surface will not affect the surface but will instead trigger a `#CAIRO_STATUS_SURFACE_FINISHED` error.

When the last call to csurface:Free() decreases the reference count to zero, Cairo will call csurface:Finish() if it hasn't been called already, before freeing the resources associated with the surface.

INPUTS

none

14.7 csurface:Flush

NAME
csurface:Flush – flush surface

SYNOPSIS
`csurface:Flush()`

FUNCTION
Do any pending drawing for the surface and also restore any temporary modifications Cairo has made to the surface's state. This function must be called before switching from drawing on the surface with Cairo to drawing on it directly with native APIs, or accessing its memory outside of Cairo. If the surface doesn't support direct access, then this function does nothing.

INPUTS
none

14.8 csurface:Free

NAME
csurface:Free – destroy surface

SYNOPSIS
`csurface:Free()`

FUNCTION
Decreases the reference count on the surface by one. If the result is zero, then the surface and all associated resources are freed. See csurface:Reference().

INPUTS
none

14.9 csurface:GetContent

NAME
csurface:GetContent – get content type

SYNOPSIS
`content = csurface:GetContent()`

FUNCTION
This function returns the content type of the surface which indicates whether the surface contains color and/or alpha information. This can be one of the following constants:

`#CAIRO_CONTENT_COLOR`
: The surface holds color content only.

`#CAIRO_CONTENT_ALPHA`
: The surface holds alpha content only.

`#CAIRO_CONTENT_COLOR_ALPHA`
: The surface holds color and alpha content.

INPUTS
none

RESULTS

`content` the content type of the surface

14.10 csurface:GetDeviceOffset

NAME
csurface:GetDeviceOffset – get device offset

SYNOPSIS
```
x_offset, y_offset = csurface:GetDeviceOffset()
```

FUNCTION
This function returns the previous device offset set by csurface:SetDeviceOffset().

INPUTS
none

RESULTS

`x_offset` the offset in the X direction, in device units

`y_offset` the offset in the Y direction, in device units

14.11 csurface:GetDeviceScale

NAME
csurface:GetDeviceScale – get device scale

SYNOPSIS
```
x_scale, y_scale = csurface:GetDeviceScale()
```

FUNCTION
This function returns the previous device scale set by csurface:SetDeviceScale().

INPUTS
none

RESULTS

`x_scale` the scale in the X direction, in device units

`y_scale` the scale in the Y direction, in device units

14.12 csurface:GetDocumentUnit

NAME
csurface:GetDocumentUnit – get document unit

SYNOPSIS
```
unit = csurface:GetDocumentUnit()
```

FUNCTION
Get the unit of the SVG surface.

If the surface passed as an argument is not an SVG surface, the function sets the error status to `#CAIRO_STATUS_SURFACE_TYPE_MISMATCH` and returns `#CAIRO_SVG_UNIT_USER`.

INPUTS
none

RESULTS

`unit`	the SVG unit of the SVG surface

14.13 csurface:GetFallbackResolution

NAME
csurface:GetFallbackResolution – get fallback resolution

SYNOPSIS
```
xppi, yppi = csurface:GetFallbackResolution()
```

FUNCTION
This function returns the previous fallback resolution set by csurface:SetFallbackResolution(), or default fallback resolution if never set.

INPUTS
none

RESULTS

`xppi`	horizontal pixels per inch
`yppi`	vertical pixels per inch

14.14 csurface:GetFontOptions

NAME
csurface:GetFontOptions – get font options

SYNOPSIS
```
csurface:GetFontOptions(options)
```

FUNCTION
Retrieves the default font rendering options for the surface. This allows display surfaces to report the correct subpixel order for rendering on them, print surfaces to disable hinting of metrics and so forth. The result can then be used with cairo.ScaledFont().

INPUTS

`options` a Cairo font options object into which to store the retrieved options; all existing values are overwritten

14.15 csurface:GetFormat

NAME

csurface:GetFormat – get pixel format of surface

SYNOPSIS

```
format = csurface:GetFormat()
```

FUNCTION

Get the pixel format of the surface. See Section 4.5 [cairo.ImageSurface], page 16, for a list of pixel formats.

INPUTS

none

RESULTS

`format` pixel format of the surface

14.16 csurface:GetHeight

NAME

csurface:GetHeight – get surface height

SYNOPSIS

```
height = csurface:GetHeight()
```

FUNCTION

Get the height of the image surface in pixels.

INPUTS

none

RESULTS

`height` the height of the surface in pixels

14.17 csurface:GetMimeData

NAME

csurface:GetMimeData – get mime data

SYNOPSIS

```
data$ = csurface:GetMimeData(mime_type$)
```

FUNCTION

Return mime data previously attached to the surface using the specified mime type. If no data has been attached with the given mime type, `data$` is set to `Nil`.

INPUTS

`mime_type$`
: the mime type of the image data

RESULTS

`data$` the image data to attached to the surface

14.18 csurface:GetReferenceCount

NAME

csurface:GetReferenceCount – get reference count

SYNOPSIS

```
count = csurface:GetReferenceCount()
```

FUNCTION

Returns the current reference count of the surface.

INPUTS

none

RESULTS

`count` reference count of the surface

14.19 csurface:GetType

NAME

csurface:GetType – get surface type

SYNOPSIS

```
type = csurface:GetType()
```

FUNCTION

This function returns the type of the backend used to create a surface. This can be one of the following types:

`#CAIRO_SURFACE_TYPE_IMAGE`
: The surface is of type image.

`#CAIRO_SURFACE_TYPE_PDF`
: The surface is of type pdf.

`#CAIRO_SURFACE_TYPE_PS`
: The surface is of type ps.

`#CAIRO_SURFACE_TYPE_SVG`
: The surface is of type svg.

`#CAIRO_SURFACE_TYPE_QUARTZ`
: The surface is of type quartz.

`#CAIRO_SURFACE_TYPE_QUARTZ_IMAGE`
: The surface is of type quartz image.

`#CAIRO_SURFACE_TYPE_WIN32`
: The surface is of type win32.

INPUTS
none

RESULTS

`type`	the type of the surface (see above)

14.20 csurface:GetWidth

NAME
csurface:GetWidth – get surface width

SYNOPSIS
```
width = csurface:GetWidth()
```

FUNCTION
Get the width of the image surface in pixels.

INPUTS
none

RESULTS

`width`	the width of the surface in pixels

14.21 csurface:IsNull

NAME
csurface:IsNull – check if surface is invalid

SYNOPSIS
```
bool = csurface:IsNull()
```

FUNCTION
Returns `True` if the surface is `NULL`, i.e. invalid. If functions that allocate surfaces fail, they might not throw an error but simply set the surface to `NULL`. You can use this function to check if surface allocation has failed in which case the surface will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS
none

RESULTS

`bool`	`True` if the surface is `NULL`, otherwise `False`

14.22 csurface:MarkDirty

NAME
csurface:MarkDirty – mark surface dirty

SYNOPSIS
```
csurface:MarkDirty()
```

FUNCTION
Tells Cairo that drawing has been done to surface using means other than Cairo, and that Cairo should reread any cached areas. Note that you must call csurface:Flush() before doing such drawing.

INPUTS
none

14.23 csurface:MarkDirtyRectangle

NAME
csurface:MarkDirtyRectangle – mark dirty rectangle

SYNOPSIS
```
csurface:MarkDirtyRectangle(x, y, width, height)
```

FUNCTION
Like csurface:MarkDirty(), but drawing has been done only to the specified rectangle, so that Cairo can retain cached contents for other parts of the surface.

Any cached clip set on the surface will be reset by this function, to make sure that future Cairo calls have the clip set that they expect.

INPUTS

`x`	X coordinate of dirty rectangle
`y`	Y coordinate of dirty rectangle
`width`	width of dirty rectangle
`height`	height of dirty rectangle

14.24 csurface:Reference

NAME
csurface:Reference – increase reference count

SYNOPSIS
```
csurface:Reference()
```

FUNCTION
Increases the reference count on the surface by one. This prevents the surface from being destroyed until a matching call to csurface:Free() is made.

Use csurface:GetReferenceCount() to get the number of references to a surface.

INPUTS
none

14.25 csurface:RestrictToVersion

NAME
csurface:RestrictToVersion – restrict to version

SYNOPSIS
`csurface:RestrictToVersion(version)`

FUNCTION
Restricts the generated SVG or PDF file to `version`. For PDF surfaces, you have to pass a PDF version constant here. See cairo.PDFSurface() for a list of available PDF versions. For SVG surfaces, you have to pass an SVG version constant here. See cairo.SVGSurface() for a list of available SVG versions.

This function should only be called before any drawing operations have been performed on the given surface. The simplest way to do this is to call this function immediately after creating the surface.

INPUTS

`version` PDF or SVG version (see above)

14.26 csurface:SetDeviceOffset

NAME
csurface:SetDeviceOffset – set device offset

SYNOPSIS
`csurface:SetDeviceOffset(x_offset, y_offset)`

FUNCTION
Sets an offset that is added to the device coordinates determined by the CTM when drawing to the surface. One use case for this function is when we want to create a surface that redirects drawing for a portion of an onscreen surface to an offscreen surface in a way that is completely invisible to the user of the Cairo API. Setting a transformation via ccontext:Translate() isn't sufficient to do this, since functions like ccontext:DeviceToUser() will expose the hidden offset.

Note that the offset affects drawing to the surface as well as using the surface in a source pattern.

INPUTS

`x_offset` the offset in the X direction, in device units

`y_offset` the offset in the Y direction, in device units

14.27 csurface:SetDeviceScale

NAME

csurface:SetDeviceScale – set device scale

SYNOPSIS

```
csurface:SetDeviceScale(x_scale, y_scale)
```

FUNCTION

Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to the surface. One common use for this is to render to very high resolution display devices at a scale factor, so that code that assumes 1 pixel will be a certain size will still work. Setting a transformation via ccontext:Scale() isn't sufficient to do this, since functions like ccontext:DeviceToUser() will expose the hidden scale.

Note that the scale affects drawing to the surface as well as using the surface in a source pattern.

INPUTS

`x_scale`	a scale factor in the X direction
`y_scale`	a scale factor in the Y direction

14.28 csurface:SetDocumentUnit

NAME

csurface:SetDocumentUnit – set document unit

SYNOPSIS

```
csurface:SetDocumentUnit(unit)
```

FUNCTION

Use the specified unit for the width and height of the generated SVG file. See csurface:GetDocumentUnit() for a list of available unit values that can be used here.

This function can be called at any time before generating the SVG file.

However to minimize the risk of ambiguities it's recommended to call it before any drawing operations have been performed on the given surface, to make it clearer what the unit used in the drawing operations is.

The simplest way to do this is to call this function immediately after creating the SVG surface.

Note if this function is never called, the default unit for SVG documents generated by Cairo will be user unit.

INPUTS

`unit`	SVG unit

14.29 csurface:SetFallbackResolution

NAME

csurface:SetFallbackResolution – set fallback resolution

SYNOPSIS

```
csurface:SetFallbackResolution(x_pixels_per_inch, y_pixels_per_inch)
```

FUNCTION

Set the horizontal and vertical resolution for image fallbacks.

When certain operations aren't supported natively by a backend, Cairo will fallback by rendering operations to an image and then overlaying that image onto the output. For backends that are natively vector-oriented, this function can be used to set the resolution used for these image fallbacks, (larger values will result in more detailed images, but also larger file sizes).

Some examples of natively vector-oriented backends are the ps, pdf, and svg backends.

For backends that are natively raster-oriented, image fallbacks are still possible, but they are always performed at the native device resolution. So this function has no effect on those backends.

Note: The fallback resolution only takes effect at the time of completing a page (with ccontext:ShowPage() or ccontext:CopyPage()) so there is currently no way to have more than one fallback resolution in effect on a single page.

The default fallback resolution is 300 pixels per inch in both dimensions.

INPUTS

`x_pixels_per_inch`
: horizontal setting for pixels per inch

`y_pixels_per_inch`
: vertical setting for pixels per inch

14.30 csurface:SetMetadata

NAME

csurface:SetMetadata – set metadata for PDF surface

SYNOPSIS

```
csurface:SetMetadata(metadata, s$)
```

FUNCTION

Set document metadata for PDF surface. The `#CAIRO_PDF_METADATA_CREATE_DATE` and `#CAIRO_PDF_METADATA_MOD_DATE` values must be in ISO-8601 format: YYYY-MM-DDThh:mm:ss. An optional timezone of the form "[+/-]hh:mm" or "Z" for UTC time can be appended. All other metadata values can be any string.

The `metadata` parameter can be one of the following constants:

`#CAIRO_PDF_METADATA_TITLE`
: The document title

`#CAIRO_PDF_METADATA_AUTHOR`
: The document author

`#CAIRO_PDF_METADATA_SUBJECT`
: The document subject

`#CAIRO_PDF_METADATA_KEYWORDS`
: The document keywords

`#CAIRO_PDF_METADATA_CREATOR`
: The document creator

`#CAIRO_PDF_METADATA_CREATE_DATE`
: The document creation date

`#CAIRO_PDF_METADATA_MOD_DATE`
: The document modification date

For example:

```
surface:SetMetadata(#CAIRO_PDF_METADATA_TITLE, "My Document")
surface:SetMetadata(#CAIRO_PDF_METADATA_CREATE_DATE, "2015-12-31T23:59+02:00")
```

INPUTS

`metadata` the metadata item to set (see above)

`s$` metadata value

14.31 csurface:SetMimeData

NAME

csurface:SetMimeData – set mime data

SYNOPSIS

```
status = csurface:SetMimeData(mime_type[, data$])
```

FUNCTION

Attach an image in the format `mime_type` to the surface. To remove the data from a surface, call this function without specifying the `data$` argument.

The attached image (or filename) data can later be used by backends which support it (currently: PDF, PS, SVG and Win32 Printing surfaces) to emit this data instead of making a snapshot of the the surface. This approach tends to be faster and requires less memory and disk space.

The recognized MIME types are the following: `#CAIRO_MIME_TYPE_JPEG`, `#CAIRO_MIME_TYPE_PNG`, `#CAIRO_MIME_TYPE_JP2`, `#CAIRO_MIME_TYPE_URI`, `#CAIRO_MIME_TYPE_UNIQUE_ID`, `#CAIRO_MIME_TYPE_JBIG2`, `#CAIRO_MIME_TYPE_JBIG2_GLOBAL`, `#CAIRO_MIME_TYPE_JBIG2_GLOBAL_ID`, `#CAIRO_MIME_TYPE_CCITT_FAX`, `#CAIRO_MIME_TYPE_CCITT_FAX_PARAMS`.

See corresponding backend surface docs for details about which MIME types it can handle. Caution: the associated MIME data will be discarded if you draw on the surface afterwards. Use this function with care.

Even if a backend supports a MIME type, that does not mean Cairo will always be able to use the attached MIME data. For example, if the backend does not natively support the compositing operation used to apply the MIME data to the backend. In that case, the MIME data will be ignored. Therefore, to apply an image in all cases, it is best to create an image surface which contains the decoded image data and then attach the MIME data to that. This ensures the image will always be used while still allowing the MIME data to be used whenever possible.

This function returns `#CAIRO_STATUS_SUCCESS` or `#CAIRO_STATUS_NO_MEMORY` if a slot could not be allocated for the user data.

INPUTS

`mime_type`
: the MIME type of the image data (see above)

`data$`
: optional: the image data to attach to the surface

RESULTS

`status`
: a Cairo status code

14.32 csurface:SetPageLabel

NAME

csurface:SetPageLabel – set page label

SYNOPSIS

```
csurface:SetPageLabel(s$)
```

FUNCTION

Set page label for the current page of a PDF surface.

INPUTS

`s$`
: the desired page label

14.33 csurface:SetSize

NAME

csurface:SetSize – set PDF surface size

SYNOPSIS

```
csurface:SetSize(width_in_points, height_in_points)
```

FUNCTION

Changes the size of a PDF surface for the current (and subsequent) pages.

This function should only be called before any drawing operations have been performed on the current page. The simplest way to do this is to call this function immediately after creating the surface or immediately after completing a page with either ccontext:ShowPage() or ccontext:CopyPage().

INPUTS

`width_in_points`
: new surface width, in points (1 point == 1/72.0 inch)

`height_in_points`
: new surface height, in points (1 point == 1/72.0 inch)

14.34 csurface:SetThumbnailSize

NAME

csurface:SetThumbnailSize – set thumbnail size

SYNOPSIS

```
csurface:SetThumbnailSize(width, height)
```

FUNCTION

Set the thumbnail image size for the current and all subsequent pages of a PDF surface. Setting a width or height of 0 disables thumbnails for the current and subsequent pages.

INPUTS

`width` thumbnail width

`height` thumbnail height

14.35 csurface:ShowPage

NAME

csurface:ShowPage – show page

SYNOPSIS

```
csurface:ShowPage()
```

FUNCTION

Emits and clears the current page for backends that support multiple pages. Use csurface:CopyPage() if you don't want to clear the page.

There is a convenience function for this that takes a Cairo context, namely ccontext:ShowPage().

INPUTS

none

14.36 csurface:Status

NAME

csurface:Status – get surface status

SYNOPSIS

```
status = csurface:Status()
```

FUNCTION

Checks whether an error has previously occurred for this surface.

This function returns `#CAIRO_STATUS_SUCCESS`, `#CAIRO_STATUS_NULL_POINTER`, `#CAIRO_STATUS_NO_MEMORY`, `#CAIRO_STATUS_READ_ERROR`, `#CAIRO_STATUS_INVALID_CONTENT`, `#CAIRO_STATUS_INVALID_FORMAT`, or `#CAIRO_STATUS_INVALID_VISUAL`.

INPUTS

none

RESULTS

`status` a Cairo status value (see above)

14.37 csurface:SupportsMimeType

NAME

csurface:SupportsMimeType – supports mime type

SYNOPSIS

```
ok = csurface:SupportsMimeType(mime_type)
```

FUNCTION

Return whether the surface supports `mime_type`. See csurface:SetMimeData() for a list of MIME types.

This function returns `True` if the surface supports `mime_type`, `False` otherwise.

INPUTS

`mime_type`
the mime type

RESULTS

`ok` boolean indicating whether the surface supports `mime_type`

14.38 csurface:ToBrush

NAME

csurface:ToBrush – convert surface to Hollywood brush

SYNOPSIS

```
[id] = csurface:ToBrush(brush[, color])
```

FUNCTION

Converts the image surface to the Hollywood brush specified by `brush`. If you pass `Nil` in `brush`, automatic id selection will be used and this function will return the identifier of the brush. The optional argument `color` is only used in case the image surface doesn't have any color channels (e.g. if the pixel format is `#CAIRO_FORMAT_A8`). In that case, the color channels of the brush will be filled with the color specified in `color`.

INPUTS

`brush` identifier of the brush to be created or `Nil` for auto id selection

`color` optional: filling color to use if the surface doesn't have any color channels

RESULTS

`id` optional: handle to the new brush; will only be returned if you specified `Nil` in `brush`

14.39 csurface:WriteToPNG

NAME

csurface:WriteToPNG – write surface to PNG image

SYNOPSIS

```
status = csurface:WriteToPNG(filename$)
```

FUNCTION

Writes the contents of the surface to a new file specified by `filename$` as a PNG image

This function returns `#CAIRO_STATUS_SUCCESS` if the PNG file was written successfully. Otherwise, `#CAIRO_STATUS_NO_MEMORY` if memory could not be allocated for the operation or `#CAIRO_STATUS_SURFACE_TYPE_MISMATCH` if the surface does not have pixel contents, or `#CAIRO_STATUS_WRITE_ERROR` if an I/O error occurs while attempting to write the file, or `#CAIRO_STATUS_PNG_ERROR` if libpng returned an error.

INPUTS

`filename$`
filename of the PNG image

RESULTS

`status` a Cairo status value (see above)

15 Pango functions

15.1 pango.Attribute

NAME

pango.Attribute – create Pango attribute

SYNOPSIS

```
attr = pango.Attribute(type$[, ...])
```

FUNCTION

Creates a new Pango attribute of the specified `type$`. The other parameters that must be passed to this function depend on the specified type.

The following types are currently supported:

`AllowBreaks`
: Create a new allow-breaks attribute. You must pass an additional boolean argument specifying if breaks are allowed. If breaks are disabled, the range will be kept in a single run, as far as possible.

`Background`
: Create a new background color attribute. You must pass the three additional arguments which specify the red, green, and blue components of the background color and must be between 0 and 65535 each.

`BackgroundAlpha`
: Create a new background alpha attribute. You must pass the additional argument `alpha` which specifies the background alpha value and must be between 0 and 65535.

`Fallback`
: Create a new font fallback attribute. You must pass an additional boolean argument that specifies whether fallback should be enabled or disabled. If fallback is disabled, characters will only be used from the closest matching font on the system. No fallback will be done to other fonts on the system that might contain the characters in the text.

`Family`
: Create a new font family attribute. You must pass a string that contains the family name or a comma-separated list of families.

`FontDesc`
: Create a new font description attribute. You must a Pango font description object as an additional argument. This attribute allows setting the attributes `Family`, `Style`, `Weight`, `Variant`, `Stretch`, and `Size` simultaneously.

`Foreground`
: Create a new foreground color attribute. You must pass the three additional arguments which specify the red, green, and blue components of the foreground color and must be between 0 and 65535 each.

`ForegroundAlpha`
: Create a new foreground alpha attribute. You must pass an additional value which specifies the foreground alpha value and must be between 0 and 65535.

`Gravity` Create a new gravity attribute. You must pass an additional argument that specifies the gravity. See pcontext:SetBaseGravity() for a list of gravity constants.

`GravityHint`
Create a new gravity hint attribute. You must pass an additional argument that specifies the gravity hint. See pcontext:SetGravityHint() for a list of gravity hints.

`InsertHyphens`
Create a new insert-hyphens attribute. Pango will insert hyphens when breaking lines in the middle of a word. Set this attribute to `False` to suppress the hyphen.

`Language` Create a new language tag attribute. You must pass an additional argument containing a Pango language object.

`LetterSpacing`
Create a new letter-spacing attribute. You must pass an additional argument specifying the amount of extra space to add between graphemes of the text (in Pango units).

`Rise` Create a new baseline displacement attribute. You must pass an additional argument specifying the amount that the text should be displaced vertically, in Pango units. Positive values displace the text upwards.

`Scale` Create a new font size scale attribute. The base font for the affected text will have its size multiplied by the scale factor that is passed as an additional argument to this attribute.

`Shape` Create a new shape attribute. A shape is used to impose a particular ink and logical rectangle on the result of shaping a particular glyph. This might be used, for instance, for embedding a picture or a widget inside a Pango layout. You have to pass two additional table arguments: The first is the ink rectangle to assign to each character, the second is the logical rectangle to assign to each character. Both arguments must be tables with the fields `x`, `y`, `width`, and `height` initialized. Additionally, you can pass an optional third argument which is considered user data and can be of any type.

`Show` Create a new attribute that influences how invisible characters are rendered. You must pass an additional argument that can be set to one of the following flags:

`#PANGO_SHOW_NONE`
No special treatment for invisible characters.

`#PANGO_SHOW_SPACES`
Render spaces, tabs and newlines visibly.

`#PANGO_SHOW_LINE_BREAKS`
Render line breaks visibly.

`#PANGO_SHOW_IGNORABLES`
Render default-ignorable Unicode characters visibly.

`Size` Create a new font-size attribute in fractional points. You must pass an additional argument specifying the font size, in #PANGO_SCALE-ths of a point.

`SizeAbsolute`
Create a new font-size attribute in device points. You must pass an additional argument specifying the font size, in #PANGO_SCALE-ths of a device unit.

`Stretch` Create a new font stretch attribute. You must pass an additional argument specifying the font stretch mode. See pfontdesc:SetStretch() for a list of Pango stretch modes.

`Strikethrough`
Create a new strike-through attribute. You must pass an additional boolean argument specifying if the text should be struck-through.

`StrikethroughColor`
Create a new strikethrough color attribute. You must pass the three additional arguments `red`, `green`, `blue` which specify the strikethrough color and must be between 0 and 65535 for each value.

`Style` Create a new font slant style attribute. You must pass an additional argument specifying the font slant style. See pfontdesc:SetStyle() for a list of Pango styles.

`Underline`
Create a new underline-style attribute. You must pass an additional argument specifying the underline style. The following underline styles are currently supported:

`#PANGO_UNDERLINE_NONE`
No underline should be drawn.

`#PANGO_UNDERLINE_SINGLE`
A single underline should be drawn.

`#PANGO_UNDERLINE_DOUBLE`
A double underline should be drawn.

`#PANGO_UNDERLINE_LOW`
A single underline should be drawn at a position beneath the ink extents of the text being underlined. This should be used only for underlining single characters, such as for keyboard accelerators. `#PANGO_UNDERLINE_SINGLE` should be used for extended portions of text.

`#PANGO_UNDERLINE_ERROR`
An underline indicating an error should be drawn below. The exact style of rendering is up to the PangoRenderer in use, but typical styles include wavy or dotted lines. This underline is typically used to indicate an error such as a possible mispelling; in some cases a contrasting color may automatically be used.

`#PANGO_UNDERLINE_SINGLE_LINE`
: Like PANGO_UNDERLINE_SINGLE, but drawn continuously across multiple runs.

`#PANGO_UNDERLINE_DOUBLE_LINE`
: Like PANGO_UNDERLINE_DOUBLE, but drawn continuously across multiple runs.

`#PANGO_UNDERLINE_ERROR_LINE`
: Like PANGO_UNDERLINE_ERROR, but drawn continuously across multiple runs.

`UnderlineColor`
: Create a new underline color attribute. You must pass the three additional arguments `red`, `green`, `blue` which specify the underline color and must be between 0 and 65535 for each value.

`Variant`
: Create a new font variant attribute. You must pass an additional argument specifying the font variant style. See pfontdesc:SetVariant() for a list of Pango variants.

`Weight`
: Create a new font weight attribute. You must pass an additional argument specifying the font weight. This can be a number or one of the predefined font weights. See pfontdesc:SetWeight() for a list of Pango font weights.

This function returns the newly allocated Pango attribute, which should be freed with pattribute:Free().

INPUTS

none

RESULTS

`attr` the newly allocated Pango attribute

15.2 pango.AttrList

NAME

pango.AttrList – create attribute list

SYNOPSIS

```
handle = pango.AttrList()
```

FUNCTION

Create a new empty attribute list with a reference count of one.

This function returns the newly allocated Pango attribute list, which should be freed with pattrlist:Free()

INPUTS

none

RESULTS

`handle` the newly allocated attribute list

15.3 pango.Context

NAME

pango.Context – create Pango context

SYNOPSIS

```
handle = pango.Context()
```

FUNCTION

Creates a new Pango context initialized to default values.

This function is not particularly useful as it should always be followed by a pcontext:SetFontMap() call, and the function pfontmap:CreateContext() does these two steps together and hence users are recommended to use that.

This function returns the newly allocated Pango context, which should be freed with pcontext:Free().

INPUTS

none

RESULTS

`handle` the newly allocated Pango context

15.4 pango.Coverage

NAME

pango.Coverage – create Pango coverage

SYNOPSIS

```
handle = pango.Coverage()
```

FUNCTION

Create a new Pango coverage.

This function returns the newly allocated Pango coverage, initialized to `#PANGO_COVERAGE_NONE` with a reference count of one, which should be freed with pcoverage:Free().

INPUTS

none

RESULTS

`handle` the newly allocated Pango coverage

15.5 pango.ExtentsToPixels

NAME

pango.ExtentsToPixels – convert extents from Pango to device units

SYNOPSIS

```
r1, r2 = pango.ExtentsToPixels(inclusive, nearest)
```

FUNCTION

Converts extents from Pango units to device units. The conversion is done by dividing by the `#PANGO_SCALE` factor and performing rounding. The `inclusive` and `nearest` arguments must be tables that have the fields `x`, `y`, `width`, and `height` initialized. If you don't need one of the rectangles, you can also set it to `Nil`.

The `inclusive` rectangle is converted by flooring the x/y coordinates and extending width/height, such that the final rectangle completely includes the original rectangle.

The `nearest` rectangle is converted by rounding the coordinates of the rectangle to the nearest device unit (pixel).

The rule to which argument to use is: if you want the resulting device-space rectangle to completely contain the original rectangle, pass it in as `inclusive`. If you want two touching-but-not-overlapping rectangles stay touching-but-not-overlapping after rounding to device units, pass them in as `nearest`.

INPUTS

`inclusive`
: rectangle to round to pixels inclusively or `Nil`

`nearest`
: rectangle to round to nearest pixels or `Nil`

RESULTS

`r1`
: rounded inclusive rectangle

`r2`
: rounded nearest rectangle

15.6 pango.FontDescription

NAME

pango.FontDescription – create font description

SYNOPSIS

```
desc = pango.FontDescription([s$])
```

FUNCTION

Creates a new font description. If the `s$` parameter is omitted, all fields will be initialized to default values. Otherwise, the font description is initialized according to the string specification. If specified, `s$` must have the form

```
[FAMILY_LIST] [STYLE_OPTIONS] [SIZE] [VARIATIONS]
```

where `FAMILY_LIST` is a comma-separated list of families optionally terminated by a comma, `STYLE_OPTIONS` is a whitespace-separated list of words where each word describes one of style, variant, weight, stretch, or gravity, and `SIZE` is a decimal number (size in points) or optionally followed by the unit modifier `"px"` for absolute size. `VARIATIONS` is a comma-separated list of font variation specifications of the form `"axis=value"` (the = sign is optional).

The following words are understood as styles: `"Normal"`, `"Roman"`, `"Oblique"`, `"Italic"`.

The following words are understood as variants: `"Small-Caps"`, `"All-Small-Caps"`, `"Petite-Caps"`, `"All-Petite-Caps"`, `"Unicase"`, `"Title-Caps"`.

The following words are understood as weights: "Thin", "Ultra-Light", "Extra-Light", "Light", "Semi-Light", "Demi-Light", "Book", "Regular", "Medium", "Semi-Bold", "Demi-Bold", "Bold", "Ultra-Bold", "Extra-Bold", "Heavy", "Black", "Ultra-Black", "Extra-Black".

The following words are understood as stretch values: "Ultra-Condensed", "Extra-Condensed", "Condensed", "Semi-Condensed", "Semi-Expanded", "Expanded", "Extra-Expanded", "Ultra-Expanded".

The following words are understood as gravity values: "Not-Rotated", "South", "Upside-Down", "North", "Rotated-Left", "East", "Rotated-Right", "West".

Any one of the options may be absent. If `FAMILY_LIST` is absent, then the `FamilyName` field of the resulting font description will be initialized to `Nil`. If `STYLE_OPTIONS` is missing, then all style options will be set to the default values. If `SIZE` is missing, the size in the resulting font description will be set to 0.

A typical example:

```
Cantarell Italic Light 15 wght=200
```

INPUTS

`s$` optional: a string describing the font to load

RESULTS

`desc` a new Pango font description

15.7 pango.FontMap

NAME

pango.FontMap – create font map

SYNOPSIS

```
handle = pango.FontMap([fonttype])
```

FUNCTION

Creates a new fontmap object.

A fontmap is used to cache information about available fonts, and holds certain global parameters such as the resolution. In most cases, you can use pango.GetDefaultFontMap() instead.

Note that the type of the returned object will depend on the particular font backend Cairo was compiled to use; alternatively, you can also pass the optional `fonttype` argument to explicitly specify a backend to use. See cfontface:GetType() for a list of font types.

This function returns the newly allocated Pango font map, which should be freed with pfontmap:Free().

INPUTS

`fonttype` optional: the Cairo font type to use

RESULTS

`handle` the newly allocated Pango font map

15.8 pango.GetDefaultFontMap

NAME
pango.GetDefaultFontMap – get default font map

SYNOPSIS
```
handle = pango.GetDefaultFontMap()
```

FUNCTION
Gets a default font map to use with Cairo.

Note that the type of the returned object will depend on the particular font backend Cairo was compiled to use.

The default Cairo fontmap can be changed by using pango.SetDefaultFontMap(). This can be used to change the Cairo font backend that the default fontmap uses for example.

The object returned is owned by Pango and must not be freed.

INPUTS
none

RESULTS
`handle` the default Pango fontmap

15.9 pango.GetDefaultLanguage

NAME
pango.GetDefaultLanguage – get default language

SYNOPSIS
```
lang = pango.GetDefaultLanguage()
```

FUNCTION
Returns the Pango language for the current locale of the process. Note that the default language can change over the life of an application.

INPUTS
none

RESULTS
`lang` the default language as a Pango language

15.10 pango.GlyphString

NAME
pango.GlyphString – create Pango glyph string

SYNOPSIS
```
gstr = pango.GlyphString()
```

FUNCTION

Create a new Pango glyph string.

This function returns the newly allocated Pango glyph string, which should be freed with pglyphstring:Free().

INPUTS

none

RESULTS

`gstr`	the newly allocated Pango glyph string

15.11 pango.GravityForMatrix

NAME

pango.GravityForMatrix – find gravity for matrix

SYNOPSIS

```
gravity = pango.GravityForMatrix(matrix)
```

FUNCTION

Finds the gravity that best matches the rotation component in a Pango matrix.

This function returns the gravity of `matrix`, which will never be `#PANGO_GRAVITY_AUTO`.

See pcontext:SetBaseGravity() for a list of gravity constants.

INPUTS

`matrix`	Pango matrix

RESULTS

`gravity`	the gravity of `matrix`

15.12 pango.GravityForScript

NAME

pango.GravityForScript – get gravity for script

SYNOPSIS

```
gravity = pango.GravityForScript(script, base_gravity, hint)
```

FUNCTION

Returns the gravity to use in laying out a Pango item. The gravity is determined based on the script, base gravity, and hint.

If `base_gravity` is `#PANGO_GRAVITY_AUTO`, it is first replaced with the preferred gravity of `script`. To get the preferred gravity of a script, pass `#PANGO_GRAVITY_AUTO` and `#PANGO_GRAVITY_HINT_STRONG` in.

This function returns resolved gravity suitable to use for a run of text with `script`.

See pcontext:SetBaseGravity() for a list of gravity constants. See pcontext:SetGravityHint() for a list of gravity hints. See planguage:GetScripts() for a list of scripts.

INPUTS

`script` the Pango script to query

`base_gravity`
base gravity of the paragraph

`hint` orientation hint

RESULTS

`gravity` resolved gravity suitable to use for a run of text

15.13 pango.GravityForScriptAndWidth

NAME

pango.GravityForScriptAndWidth – get gravity for script and width

SYNOPSIS

```
g = pango.GravityForScriptAndWidth(script, wide, base_gravity, hint)
```

FUNCTION

Returns the gravity to use in laying out a single character or Pango item.

The gravity is determined based on the script, East Asian width, base gravity, and hint,

This function is similar to pango.GravityForScript() except that this function makes a distinction between narrow/half-width and wide/full-width characters also. Wide/full-width characters always stand *upright*, that is, they always take the base gravity, whereas narrow/full-width characters are always rotated in vertical context.

If `base_gravity` is `#PANGO_GRAVITY_AUTO`, it is first replaced with the preferred gravity of `script`.

This function returns resolved gravity suitable to use for a run of text with `script` and `wide`.

See pcontext:SetBaseGravity() for a list of gravity constants. See pcontext:SetGravityHint() for a list of gravity hints. See planguage:GetScripts() for a list of scripts.

INPUTS

`script` the Pango script to query

`wide` `True` for wide characters

`base_gravity`
base gravity of the paragraph

`hint` orientation hint

RESULTS

`g` resolved gravity suitable to use for a run of text

15.14 pango.GravityToRotation

NAME

pango.GravityToRotation – convert gravity to rotation

SYNOPSIS

```
angle = pango.GravityToRotation(gravity)
```

FUNCTION

Converts a Pango gravity value to its natural rotation in radians.

Note that pmatrix:Rotate() takes angle in degrees, not radians. So, to call pmatrix:Rotate() with the output of this function you should multiply it by (180. / #PI).

This function returns the rotation value corresponding to `gravity`.

See pcontext:SetBaseGravity() for a list of gravity constants.

INPUTS

`gravity`	gravity to query, should not be `#PANGO_GRAVITY_AUTO`

RESULTS

`angle`	the rotation value corresponding to `gravity`

15.15 pango.Item

NAME

pango.Item – create Pango item

SYNOPSIS

```
item = pango.Item()
```

FUNCTION

Creates a new Pango item structure initialized to default values.

This function returns the newly allocated Pango item, which should be freed with pitem:Free().

INPUTS

none

RESULTS

`item`	the newly allocated Pango item

15.16 pango.Language

NAME

pango.Language – make language from string

SYNOPSIS

```
lang = pango.Language(tag$)
```

FUNCTION

Convert a language tag specified by `tag$` to a Pango language object. The language tag must be in a RFC-3066 format. Pango language handles can be efficiently copied (copy the handle) and compared with other language tags (compare the handle.)

This function first canonicalizes the string by converting it to lowercase, mapping '_' to '-', and stripping all characters other than letters and '-'.

Use pango.GetDefaultLanguage() if you want to get the Pango language for the current locale of the process.

INPUTS

none

RESULTS

`lang`	a Pango language object

15.17 pango.Layout

NAME

pango.Layout – create Pango layout

SYNOPSIS

```
handle = pango.Layout(context)
```

FUNCTION

Create a new Pango layout object with attributes initialized to default values for a particular Pango context.

INPUTS

`context`	a Pango context

RESULTS

`handle`	the newly allocated Pango layout

15.18 pango.Matrix

NAME

pango.Matrix – create matrix

SYNOPSIS

```
m = pango.Matrix([xx, xy, yx, yy, x0, y0])
```

FUNCTION

Creates a matrix and optionally initializes its affine transformation to the coefficients specified by `xx`, `xy`, `yx`, `yy`, `x0`, `y0`. Omitted coefficients will be set to 0.

INPUTS

`xx`	optional: xx component of the affine transformation (defaults to 0)
`xy`	optional: xy component of the affine transformation (defaults to 0)

`yx`	optional: yx component of the affine transformation (defaults to 0)
`yy`	optional: yy component of the affine transformation (defaults to 0)
`x0`	optional: X translation component of the affine transformation (defaults to 0)
`y0`	optional: Y translation component of the affine transformation (defaults to 0)

RESULTS

`m`	matrix object

15.19 pango.MatrixIdentity

NAME

pango.MatrixIdentity – create identity matrix

SYNOPSIS

```
m = pango.MatrixIdentity()
```

FUNCTION

Creates a matrix and initializes its affine transformation to an identity transformation.

INPUTS

none

RESULTS

`m`	identity matrix object

15.20 pango.SetDefaultFontMap

NAME

pango.SetDefaultFontMap – set default fontmap

SYNOPSIS

```
pango.SetDefaultFontMap([fontmap])
```

FUNCTION

Sets a default fontmap to use with Cairo.

This can be used to change the Cairo font backend that the default fontmap uses for example. The old default fontmap is unreffed and the new fontmap referenced.

If you omit the `fontmap` parameter, the current default fontmap will be released and a new default fontmap will be created on demand, using pango.FontMap().

INPUTS

`fontmap`	optional: Pango fontmap to set as default

15.21 pango.SetFontconfig

NAME

pango.SetFontconfig – set Fontconfig parameter

SYNOPSIS

```
pango.SetFontconfig(parm$, val$[, ...])
```

FUNCTION

This function can be used to configure individual Fontconfig settings. Fontconfig is used by Pango for font management. The following Fontconfig settings can currently be configured and passed as `parm$`:

`FontDir` Adds the specified directory to the list of directories scanned for fonts by Fontconfig. If you pass `True` as a third argument to this function, the existing list of font directories will be cleared so that `val$` is the only directory where Fontconfig will be looking for fonts. If you omit the third argument or set it to `False`, the specified directory will be added on top of the existing font directories.

`CacheDir` Sets the cache directory used by Fontconfig.

`ConfigDir`
Sets the directory in which Fontconfig looks for and stores configuration files.

`ConfigFile`
Sets the configuration file that Fontconfig should use.

INPUTS

`parm$`	setting to modify (see above for valid types)
`val$`	new value for the Fontconfig setting
`...`	optional: further parameters depending on the type passed in `parm$`

15.22 pango.Shape

NAME

pango.Shape – convert text to glyphs

SYNOPSIS

```
pango.Shape(text$, analysis, glyphs)
```

FUNCTION

Convert the characters in `text$` into glyphs.

Given a segment of text and the corresponding Pango analysis structure returned from pcontext:Itemize(), convert the characters into glyphs. You may also pass in only a substring of the item from pcontext:Itemize().

It is recommended that you use pango.ShapeFull() instead, since that API allows for shaping interaction happening across text item boundaries.

Note that the extra attributes in the `analyis` that is returned from pcontext:Itemize() have indices that are relative to the entire paragraph, so you need to subtract the item offset from their indices before calling this function.

INPUTS

`text$`	the text to process
`analysis`	Pango analysis object from pcontext:Itemize()
`glyphs`	Pango glyph string object in which to store results

15.23 pango.ShapeFull

NAME

pango.ShapeFull – convert text to glyphs

SYNOPSIS

```
pango.ShapeFull(item_text$, paragraph_text$, analysis, glyphs[, flags])
```

FUNCTION

Convert the characters in `item_text$` into glyphs.

Given a segment of text and the corresponding Pango analysis structure returned from pcontext:Itemize(), convert the characters into glyphs. You may also pass in only a substring of the item from pcontext:Itemize().

This is similar to pango.Shape(), except it also can optionally take the full paragraph text as input, which will then be used to perform certain cross-item shaping interactions. If you have access to the broader text of which `item_text$` is part of, provide the broader text as `paragraph_text$`. If `paragraph_text$` is `Nil`, item text is used instead.

Note that the extra attributes in the `analyis` that is returned from pcontext:Itemize() have indices that are relative to the entire paragraph, so you do not pass the full paragraph text as `paragraph_text$`, you need to subtract the item offset from their indices before calling pcontext:Itemize().

The optional argument `flags` can be used to influence the shaping process. The following flags are currently recognized:

`#PANGO_SHAPE_NONE`
: Default value.

`#PANGO_SHAPE_ROUND_POSITIONS`
: Round glyph positions and widths to whole device units. This option should be set if the target renderer can't do subpixel positioning of glyphs.

INPUTS

`item_text$`
: text to shape

`paragraph_text$`
: text of the paragraph or `Nil` (see details)

`analysis`	Pango analysis object from pcontext:Itemize()
`glyphs`	glyph string in which to store results
`flags`	optional: additional flags (see above)

15.24 pango.TabArray

NAME

pango.TabArray – create tab array

SYNOPSIS

```
tabs = pango.TabArray(initial_size, positions_in_pixels)
```

FUNCTION

Creates an array of `initial_size` tab stops. Tab stops are specified in pixel units if `positions_in_pixels` is `True`, otherwise in Pango units. All stops are initially at position 0.

This function returns the newly allocated Pango tab array, which should be freed with ptabarray:Free().

INPUTS

`initial_size`
: initial number of tab stops to allocate, can be 0

`positions_in_pixels`
: whether positions are in pixel units

RESULTS

`tabs` the newly allocated Pango tab array

15.25 pango.TabArrayWithPositions

NAME

pango.TabArrayWithPositions – create tab array

SYNOPSIS

```
tabs = pango.TabArrayWithPositions(positions_in_pixels[, align1, pos1, ...])
```

FUNCTION

Creates a Pango tab array and allows you to specify the alignment and position of each tab stop. Tab stops are specified in pixel units if `positions_in_pixels` is `True`, otherwise in Pango units.

For each tab stop you must provide an alignment and a position. The `alignN` parameter can be one of the following constants:

`#PANGO_TAB_LEFT`
: The text appears to the right of the tab stop position.

`#PANGO_TAB_RIGHT`
: The text appears to the left of the tab stop position until the available space is filled.

`#PANGO_TAB_CENTER`
: The text is centered at the tab stop position until the available space is filled.

`#PANGO_TAB_DECIMAL`
: Text before the first occurrence of the decimal point character appears to the left of the tab stop position (until the available space is filled), the rest to the right.

This function returns the newly allocated Pango tab array, which should be freed with ptabarray:Free().

INPUTS

`positions_in_pixels`
: whether positions are in pixel units

`align1` alignment of first tab stop (see above)

`pos1` position of first tab stop

`...` additional alignment/position pairs

RESULTS

`tabs` the newly allocated Pango tab array

15.26 pango.Version

NAME

pango.Version – return Pango version

SYNOPSIS

```
ver$ = pango.Version()
```

FUNCTION

Returns the version of Pango available at run-time.

INPUTS

none

RESULTS

`ver$` Pango version

16 Pango analysis

16.1 panalysis:Get

NAME

panalysis:Get – get analysis info

SYNOPSIS

```
table = panalysis:Get()
```

FUNCTION

Gets information about a Pango analysis object. The data will be returned as a table. The following fields will be initialized in the table:

`Font` The font for this segment. This is a Pango font handle. It is owned by the Pango analysis object and mustn't be freed.

`Level` The bidirectional level for this segment.

`Gravity` The glyph orientation for this segment. See pcontext:SetBaseGravity() for a list of gravity constants.

`Flags` Boolean flags for this segment.

`Script` The detected script for this segment. See planguage:GetScripts() for a list of supported scripts.

`Language` The detected language for this segment. This is a Pango language handle. It is owned by the Pango analysis object and mustn't be freed.

`ExtraAttrs`
Extra attributes for this segment. This is a table of Pango attributes. The Pango attribute objects are owned by the Pango analysis object and mustn't be freed.

INPUTS

none

RESULTS

`table` a table containing the analysis data

17 Pango attribute

17.1 pattribute:Copy

NAME
pattribute:Copy – copy attribute

SYNOPSIS
```
attr = pattribute:Copy()
```

FUNCTION
Make a copy of an attribute.

This function returns the newly allocated Pango attribute, which should be freed with pattribute:Free().

INPUTS
none

RESULTS

`attr`	the newly allocated attribute copy

17.2 pattribute:Equal

NAME
pattribute:Equal – compare for equality

SYNOPSIS
```
ok = pattribute:Equal(attr2)
```

FUNCTION
Compare the attribute with `attr2` for equality.

This compares only the actual value of the two attributes and not the ranges that the attributes apply to.

This function returns `True` if the two attributes have the same value.

INPUTS

`attr2`	another Pango attribute object

RESULTS

`ok`	`True` if the two attributes have the same value, `False` otherwise

17.3 pattribute:Free

NAME
pattribute:Free – free attribute

SYNOPSIS
```
pattribute:Free()
```

FUNCTION
Destroy a Pango attribute and free all associated memory.

INPUTS
none

17.4 pattribute:GetRange

NAME
pattribute:GetRange – get attribute range

SYNOPSIS
```
start_index, end_index = pattribute:GetRange()
```

FUNCTION
Gets the range to which the value in the type-specific part of the attribute applies.

INPUTS
none

RESULTS

`start_index`
: the start index of the range (in bytes)

`end_index`
: end index of the range (in bytes); the character at this index is not included in the range

17.5 pattribute:GetType

NAME
pattribute:GetType – get attribute type

SYNOPSIS
```
type = pattribute:GetType()
```

FUNCTION
Gets the attribute type. This can be one of the following constants:

```
#PANGO_ATTR_ABSOLUTE_SIZE
#PANGO_ATTR_ALLOW_BREAKS
#PANGO_ATTR_BACKGROUND
#PANGO_ATTR_BACKGROUND_ALPHA
#PANGO_ATTR_FALLBACK
#PANGO_ATTR_FAMILY
#PANGO_ATTR_FONT_DESC
#PANGO_ATTR_FONT_FEATURES
#PANGO_ATTR_FOREGROUND
#PANGO_ATTR_FOREGROUND_ALPHA
#PANGO_ATTR_GRAVITY
```

```
#PANGO_ATTR_GRAVITY_HINT
#PANGO_ATTR_INSERT_HYPHENS
#PANGO_ATTR_INVALID
#PANGO_ATTR_LANGUAGE
#PANGO_ATTR_LETTER_SPACING
#PANGO_ATTR_RISE
#PANGO_ATTR_SCALE
#PANGO_ATTR_SHAPE
#PANGO_ATTR_SHOW
#PANGO_ATTR_SIZE
#PANGO_ATTR_STRETCH
#PANGO_ATTR_STRIKETHROUGH
#PANGO_ATTR_STRIKETHROUGH_COLOR
#PANGO_ATTR_STYLE
#PANGO_ATTR_UNDERLINE
#PANGO_ATTR_UNDERLINE_COLOR
#PANGO_ATTR_VARIANT
#PANGO_ATTR_WEIGHT
```

INPUTS

none

RESULTS

`type` the attribute's type

17.6 pattribute:GetValue

NAME

pattribute:GetValue – get attribute value

SYNOPSIS

```
v = pattribute:GetValue()
```

FUNCTION

Gets the attribute value. The attribute value is the value passed to the attribute when calling pango.Attribute(). Depending on the attribute type, there can also be multiple values in the attribute.

INPUTS

none

RESULTS

`v` the attribute's value

17.7 pattribute:IsNull

NAME

pattribute:IsNull – check if attribute is invalid

SYNOPSIS

```
bool = pattribute:IsNull()
```

FUNCTION

Returns `True` if the attribute is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the attribute will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the attribute is `NULL`, otherwise `False`

17.8 pattribute:SetRange

NAME

pattribute:SetRange – set attribute range

SYNOPSIS

```
pattribute:SetRange(start_index, end_index)
```

FUNCTION

Sets the range to which the value in the type-specific part of the attribute applies.

INPUTS

`start_index`
: the start index of the range (in bytes)

`end_index`
: end index of the range (in bytes); the character at this index is not included in the range

18 Pango attribute list

18.1 pattrlist:Change

NAME

pattrlist:Change – insert attribute

SYNOPSIS

```
pattrlist:Change(attr)
```

FUNCTION

Insert the given attribute into the Pango attribute list.

It will replace any attributes of the same type on that segment and be merged with any adjoining attributes that are identical.

This function is slower than pattrlist:Insert() for creating an attribute list in order (potentially much slower for large lists). However, pattrlist:Insert() is not suitable for continually changing a set of attributes since it never removes or combines existing attributes.

INPUTS

`attr`	the attribute to insert

18.2 pattrlist:Copy

NAME

pattrlist:Copy – copy list

SYNOPSIS

```
list = pattrlist:Copy()
```

FUNCTION

Copy the attribute list and return an identical new list.

INPUTS

none

RESULTS

`list`	the newly allocated attribute list

18.3 pattrlist:Free

NAME

pattrlist:Free – free attribute list

SYNOPSIS

```
pattrlist:Free()
```

FUNCTION

Decrease the reference count of the given attribute list by one.

If the result is zero, free the attribute list and the attributes it contains.

INPUTS
none

18.4 pattrlist:GetAttributes

NAME
pattrlist:GetAttributes – get attributes

SYNOPSIS
`t = pattrlist:GetAttributes()`

FUNCTION
Gets a list of all attributes in the list. This function returns a table containing all attributes in the list. You must free the individual attributes using pattribute:Free().

INPUTS
none

RESULTS

`t`	table containing all attributes in the list

18.5 pattrlist:Insert

NAME
pattrlist:Insert – insert attribute

SYNOPSIS
`pattrlist:Insert(attr)`

FUNCTION
Insert the given attribute into the Pango attribute list.

It will be inserted after all other attributes with a matching start index. You can use pattribute:SetRange() to set the start index.

INPUTS

`attr`	the attribute to insert

18.6 pattrlist:InsertBefore

NAME
pattrlist:InsertBefore – insert attribute in before mode

SYNOPSIS
`pattrlist:InsertBefore(attr)`

FUNCTION
Insert the given attribute into the Pango attribute list.

It will be inserted before all other attributes with a matching start index. You can use pattribute:SetRange() to set the start index.

INPUTS

`attr` the attribute to insert

18.7 pattrlist:IsNull

NAME

pattrlist:IsNull – check if attribute list is invalid

SYNOPSIS

```
bool = pattrlist:IsNull()
```

FUNCTION

Returns `True` if the attribute list is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the attribute list will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the attribute list is `NULL`, otherwise `False`

18.8 pattrlist:Reference

NAME

pattrlist:Reference – increase reference count

SYNOPSIS

```
pattrlist:Reference()
```

FUNCTION

Increase the reference count of the given attribute list by one.

INPUTS

none

18.9 pattrlist:Splice

NAME

pattrlist:Splice – splice attribute list

SYNOPSIS

```
pattrlist:Splice(other, pos, len)
```

FUNCTION

This function opens up a hole in the list, fills it in with attributes from the left, and then merges the attribute list specified by `other` on top of the hole.

This operation is equivalent to stretching every attribute that applies at position `pos` in the list by an amount `len`, and then calling pattrlist:Change() with a copy of each attribute in `other` in sequence (offset in position by `pos`, and limited in length to `len`).

This operation proves useful for, for instance, inserting a pre-edit string in the middle of an edit buffer.

For backwards compatibility, the function behaves differently when `len` is 0. In this case, the attributes from `other` are not limited to `len`, and are just overlayed on top of the list.

This mode is useful for merging two lists of attributes together.

INPUTS

`other`	another Pango attribute list
`pos`	the position in the list at which to insert `other`
`len`	the length of the spliced segment; note that this must be specified since the attributes in `other` may only be present at some subsection of this range

18.10 pattrlist:Update

NAME

pattrlist:Update – update attribute indices

SYNOPSIS

```
pattrlist:Update(pos, remove, add)
```

FUNCTION

Update indices of attributes in the list for a change in the text they refer to.

The change that this function applies is removing `remove` bytes at position `pos` and inserting `add` bytes instead.

Attributes that fall entirely in the (`pos`, `pos + remove`) range are removed.

Attributes that start or end inside the (`pos`, `pos + remove`) range are shortened to reflect the removal.

Attributes start and end positions are updated if they are behind `pos + remove`.

INPUTS

`pos`	the position of the change
`remove`	the number of removed bytes
`add`	the number of added bytes

19 Pango context

19.1 pcontext:Changed

NAME
pcontext:Changed – force context change

SYNOPSIS
```
pcontext:Changed()
```

FUNCTION
Forces a change in the context, which will cause any Pango layout using this context to re-layout.

This function is only useful when implementing a new backend for Pango, something applications won't do. Backends should call this function if they have attached extra data to the context and such data is changed.

INPUTS
none

19.2 pcontext:Free

NAME
pcontext:Free – free context

SYNOPSIS
```
pcontext:Free()
```

FUNCTION
Decrease the reference count of a Pango context by one.

If the result is zero, the context and all associated memory will be freed.

INPUTS
none

19.3 pcontext:GetBaseDir

NAME
pcontext:GetBaseDir – get base direction for context

SYNOPSIS
```
dir = pcontext:GetBaseDir()
```

FUNCTION
Retrieves the base direction for the context. See pcontext:SetBaseDir() for more information.

INPUTS
none

RESULTS

`dir` the base direction for the context

19.4 pcontext:GetBaseGravity

NAME

pcontext:GetBaseGravity – get base gravity for context

SYNOPSIS

```
gravity = pcontext:GetBaseGravity()
```

FUNCTION

Retrieves the base gravity for the context. See pcontext:SetBaseGravity() for more information.

INPUTS

none

RESULTS

`gravity` the base gravity for the context

19.5 pcontext:GetFontDescription

NAME

pcontext:GetFontDescription – get font description

SYNOPSIS

```
handle = pcontext:GetFontDescription()
```

FUNCTION

Retrieve the default font description for the context.

This function returns a handle to the context's default font description. This value must not be modified or freed.

INPUTS

none

RESULTS

`handle` a handle to the context's default font

19.6 pcontext:GetFontMap

NAME

pcontext:GetFontMap – get font map

SYNOPSIS

```
handle = pcontext:GetFontMap()
```

FUNCTION

Gets the Pango font map used to look up fonts for this context.

This function returns the font map for the Pango context. This value is owned by Pango and should not be freed.

INPUTS

none

RESULTS

`handle` the font map for the Pango context

19.7 pcontext:GetFontOptions

NAME

pcontext:GetFontOptions – get font options

SYNOPSIS

```
handle = pcontext:GetFontOptions()
```

FUNCTION

Retrieves any font rendering options previously set with pcontext:SetFontOptions().

This function does not report options that are derived from the target surface by pcontext:UpdateContext().

This function returns the Cairo font options object previously set on the context, or `NULL` if no options have been set. The return object is owned by the context and must not be modified or freed. You can use cfontoptions:IsNull() to check if the returned object is `NULL`.

INPUTS

none

RESULTS

`handle` a Cairo font options object

19.8 pcontext:GetGravity

NAME

pcontext:GetGravity – get gravity

SYNOPSIS

```
gravity = pcontext:GetGravity()
```

FUNCTION

Retrieves the gravity for the context.

This is similar to pcontext:GetBaseGravity(), except for when the base gravity is `#PANGO_GRAVITY_AUTO` for which pango.GravityForMatrix() is used to return the gravity from the current context matrix.

INPUTS
none

RESULTS
`gravity` the resolved gravity for the context

19.9 pcontext:GetGravityHint

NAME
pcontext:GetGravityHint – get gravity hint

SYNOPSIS
```
hint = pcontext:GetGravityHint()
```

FUNCTION
Retrieves the gravity hint for the context. See pcontext:SetGravityHint() for details.

INPUTS
none

RESULTS
`hint` the gravity hint for the context

19.10 pcontext:GetLanguage

NAME
pcontext:GetLanguage – get language

SYNOPSIS
```
lang = pcontext:GetLanguage()
```

FUNCTION
Retrieves the global language tag for the context. The return value will be a handle to a Pango language object.

INPUTS
none

RESULTS
`lang` handle to the global language tag

19.11 pcontext:GetMatrix

NAME
pcontext:GetMatrix – get matrix

SYNOPSIS
```
m = pcontext:GetMatrix()
```

FUNCTION

Gets the transformation matrix that will be applied when rendering with this context. See pcontext:SetMatrix() for more information.

INPUTS

none

RESULTS

`m` a Pango matrix object

19.12 pcontext:GetMetrics

NAME

pcontext:GetMetrics – get metrics

SYNOPSIS

```
metrics = pcontext:GetMetrics(desc, language)
```

FUNCTION

Get overall metric information for a particular font description.

Since the metrics may be substantially different for different scripts, a language tag can be provided to indicate that the metrics should be retrieved that correspond to the script(s) used by that language.

The Pango font description is interpreted in the same way as by pcontext:Itemize(), and the family name may be a comma separated list of names. If characters from multiple of these families would be used to render the string, then the returned fonts would be a composite of the metrics for the fonts loaded for the individual families.

This function returns a Pango font metrics object. The caller must call pfontmetrics:Free() when finished using the object.

INPUTS

`desc` a Pango font description structure; `Nil` means that the font description from the context will be used

`language` language tag used to determine which script to get the metrics for; `Nil` means that the language tag from the context will be used; if no language tag is set on the context, metrics for the default language (as determined bypango.GetDefaultLanguage() will be returned

RESULTS

`metrics` a Pango font metrics object

19.13 pcontext:GetResolution

NAME

pcontext:GetResolution – get context resolution

SYNOPSIS

```
res = pcontext:GetResolution()
```

FUNCTION

Gets the resolution for the context.

This function returns the resolution in "dots per inch". A negative value will be returned if no resolution has previously been set.

See pcontext:SetResolution() for details.

INPUTS

none

RESULTS

`res` the context resolution in "dots per inch"

19.14 pcontext:GetRoundGlyphPositions

NAME

pcontext:GetRoundGlyphPositions – get round glyph positions flag

SYNOPSIS

```
ok = pcontext:GetRoundGlyphPositions()
```

FUNCTION

Returns whether font rendering with this context should round glyph positions and widths.

INPUTS

none

RESULTS

`ok` `True` or `False` indicating whether glyph positions and widths should be rounded

19.15 pcontext:GetSerial

NAME

pcontext:GetSerial – get serial of context

SYNOPSIS

```
s = pcontext:GetSerial()
```

FUNCTION

Returns the current serial number of the context.

The serial number is initialized to an small number larger than zero when a new context is created and is increased whenever the context is changed using any of the setter functions, or the Pango font map it uses to find fonts has changed. The serial may wrap, but will never have the value 0. Since it can wrap, never compare it with "less than", always use "not equals".

This can be used to automatically detect changes to a Pango context, and is only useful when implementing objects that need update when their Pango context changes, like Pango layout.

INPUTS
none

RESULTS

`s`	the current serial number of the context

19.16 pcontext:IsNull

NAME
pcontext:IsNull – check if context is invalid

SYNOPSIS
```
bool = pcontext:IsNull()
```

FUNCTION
Returns `True` if the context is `NULL`, i.e. invalid. If functions that allocate contexts fail, they might not throw an error but simply set the context to `NULL`. You can use this function to check if context allocation has failed in which case the context will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS
none

RESULTS

`bool`	`True` if the context is `NULL`, otherwise `False`

19.17 pcontext:Itemize

NAME
pcontext:Itemize – check if context is invalid

SYNOPSIS
```
list = pcontext:Itemize(text$, start_index, length, attrs[, base_dir])
```

FUNCTION
Breaks a piece of text into segments with consistent directional level and font. Each byte of text will be contained in exactly one of the items in the returned list. The generated list of items will be in logical order (the start offsets of the items are ascending).

Optionally, you can specify the base direction in the `base_dir` argument. The base direction is used when computing bidirectional levels. If `base_dir` is omitted, the base direction will be obtained from the context. See pcontext:SetBaseDir() for a list of base directions.

Note that you must free the individual Pango items in the list using pitem:Free().

INPUTS

`text$`	the text to itemize

`start_index`
first byte in text to process

`length` the number of bytes (not characters) to process after `start_index`. This must be >= 0

`attrs` a Pango attribute list containing the set of attributes that apply to text

`base_dir` optional: base direction to use for bidirectional processing

RESULTS

`list` a table containing a list of Pango items; the caller is responsible for freeing each item using pitem:Free()

19.18 pcontext:ListFamilies

NAME
pcontext:ListFamilies – list families

SYNOPSIS
```
t = pcontext:ListFamilies()
```

FUNCTION
List all families for a context. The families are returned as a table containing a list of Pango font family handles. The font family handles are owned by the context and mustn't be freed.

INPUTS
none

RESULTS

`t` table containing all font families

19.19 pcontext:LoadFont

NAME
pcontext:LoadFont – load font

SYNOPSIS
```
font = pcontext:LoadFont(desc)
```

FUNCTION
Loads the font in one of the fontmaps in the context that is the closest match for the Pango font description passed in `desc`.

This function returns the newly allocated Pango font that was loaded, or `NULL` if no font matched. You can use pfont:IsNull() to check if the returned object contains a `NULL` font.

INPUTS

`desc` a Pango font description describing the font to load

RESULTS

`font` the newly allocated Pango font

19.20 pcontext:LoadFontset

NAME

pcontext:LoadFontset – load fontset

SYNOPSIS

```
fontset = pcontext:LoadFontset(desc, language)
```

FUNCTION

Load a set of fonts in the context that can be used to render a font matching `desc`.

This function returns the newly allocated Pango font set that was loaded, or `NULL` if no font matched. You can use pfontset:IsNull() to check if the returned object contains a `NULL` font set.

INPUTS

`desc` a Pango font description describing the fonts to load

`language` a Pango language the fonts will be used for

RESULTS

`fontset` the newly allocated font set

19.21 pcontext:Reference

NAME

pcontext:Reference – increase reference count

SYNOPSIS

```
pcontext:Reference()
```

FUNCTION

Increase the reference count on the Pango context by one.

INPUTS

none

19.22 pcontext:SetBaseDir

NAME

pcontext:SetBaseDir – set base direction for context

SYNOPSIS

```
pcontext:SetBaseDir(direction)
```

FUNCTION

Sets the base direction for the context. The `direction` parameter can be set to one of the following constants:

`#PANGO_DIRECTION_LTR`
: A strong left-to-right direction.

`#PANGO_DIRECTION_RTL`
: A strong right-to-left direction.

`#PANGO_DIRECTION_WEAK_LTR`
: A weak left-to-right direction.

`#PANGO_DIRECTION_WEAK_RTL`
: A weak right-to-left direction.

`#PANGO_DIRECTION_NEUTRAL`
: No direction specified.

The base direction is used in applying the Unicode bidirectional algorithm; if the `direction` is `#PANGO_DIRECTION_LTR` or `#PANGO_DIRECTION_RTL`, then the value will be used as the paragraph direction in the Unicode bidirectional algorithm. A value of `#PANGO_DIRECTION_WEAK_LTR` or `#PANGO_DIRECTION_WEAK_RTL` is used only for paragraphs that do not contain any strong characters themselves.

INPUTS

`direction`
: the new base direction

19.23 pcontext:SetBaseGravity

NAME

pcontext:SetBaseGravity – set base gravity

SYNOPSIS

```
pcontext:SetBaseGravity(gravity)
```

FUNCTION

Sets the base gravity for the context. The `gravity` parameter can be set to one of the following constants:

`#PANGO_GRAVITY_SOUTH`
: Glyphs stand upright (default).

`#PANGO_GRAVITY_EAST`
: Glyphs are rotated 90 degrees counter-clockwise.

`#PANGO_GRAVITY_NORTH`
: Glyphs are upside-down.

`#PANGO_GRAVITY_WEST`
: Glyphs are rotated 90 degrees clockwise.

`#PANGO_GRAVITY_AUTO`
: The base gravity is used in laying vertical text out.

INPUTS

`gravity` the new base gravity (see above)

19.24 pcontext:SetFontDescription

NAME

pcontext:SetFontDescription – set font description

SYNOPSIS

```
pcontext:SetFontDescription(desc)
```

FUNCTION

Set the default font description for the context. The `desc` parameter can also be `Nil` to reset the font description to a default value.

INPUTS

`desc` the new pango font description or `Nil`

19.25 pcontext:SetFontMap

NAME

pcontext:SetFontMap – set font map

SYNOPSIS

```
pcontext:SetFontMap(font_map)
```

FUNCTION

Sets the font map to be searched when fonts are looked-up in this context. The `font_map` parameter can also be `Nil` to reset the font map to a default value.

This is only for internal use by Pango backends, a Pango context obtained via one of the recommended methods should already have a suitable font map.

INPUTS

`font_map` the Pango font map to set or `Nil`

19.26 pcontext:SetFontOptions

NAME

pcontext:SetFontOptions – set font options

SYNOPSIS

```
pcontext:SetFontOptions(options)
```

FUNCTION

Sets the font options used when rendering text with this context. These options override any options that pcontext:UpdateContext() derives from the target surface. This function will make a copy of the `options` object passed to it.

INPUTS

`options` a Cairo font obtions objects or `Nil` to unset any previously set options

19.27 pcontext:SetGravityHint

NAME

pcontext:SetGravityHint – set gravity hint

SYNOPSIS

```
pcontext:SetGravityHint(hint)
```

FUNCTION

Sets the gravity hint for the context. The `hint` parameter can be one of the following constants:

`#PANGO_GRAVITY_HINT_NATURAL`
: Scripts will take their natural gravity based on the base gravity and the script. This is the default.

`#PANGO_GRAVITY_HINT_STRONG`
: Always use the base gravity set, regardless of the script.

`#PANGO_GRAVITY_HINT_LINE`
: For scripts not in their natural direction (e.g. Latin in East gravity), choose per-script gravity such that every script respects the line progression. This means, Latin and Arabic will take opposite gravities and both flow top-to-bottom for example.

The gravity hint is used in laying vertical text out, and is only relevant if gravity of the context as returned by pcontext:GetGravity() is set to `#PANGO_GRAVITY_EAST` or `#PANGO_GRAVITY_WEST`.

INPUTS

`hint` the new gravity hint (see above)

19.28 pcontext:SetLanguage

NAME

pcontext:SetLanguage – set language

SYNOPSIS

```
pcontext:SetLanguage(language)
```

FUNCTION

Sets the global language tag for the context. The `language` parameter can also be `Nil` to reset the language to the default.

The default language for the locale of the running process can be found using pango.GetDefaultLanguage().

INPUTS

`language` the new language tag or `Nil`

19.29 pcontext:SetMatrix

NAME

pcontext:SetMatrix – set matrix

SYNOPSIS

```
pcontext:SetMatrix(matrix)
```

FUNCTION

Sets the transformation matrix that will be applied when rendering with this context.

Note that reported metrics are in the user space coordinates before the application of the matrix, not device-space coordinates after the application of the matrix. So, they don't scale with the matrix, though they may change slightly for different matrices, depending on how the text is fit to the pixel grid.

INPUTS

`matrix` a Pango matrix

19.30 pcontext:SetResolution

NAME

pcontext:SetResolution – set context resolution

SYNOPSIS

```
pcontext:SetResolution(dpi)
```

FUNCTION

Sets the resolution for the context.

This is a scale factor between points specified in a Pango font description and Cairo units. The default value is 96, meaning that a 10 point font will be 13 units high. (10 * 96. / 72. = 13.3).

Note that even though the resolution must be in "dots per inch", physical inches aren't actually involved. The terminology is conventional. A 0 or negative value means to use the resolution from the font map.

INPUTS

`dpi` the resolution in "dots per inch"

19.31 pcontext:SetRoundGlyphPositions

NAME

pcontext:SetRoundGlyphPositions – set round glyph positions

SYNOPSIS

```
pcontext:SetRoundGlyphPositions(round_positions)
```

FUNCTION

Sets whether font rendering with this context should round glyph positions and widths to integral positions, in device units.

This is useful when the renderer can't handle subpixel positioning of glyphs.

The default value is to round glyph positions, to remain compatible with previous Pango behavior.

INPUTS

`round_positions`
: boolean specifying whether to round glyph positions

19.32 pcontext:SetShapeRenderer

NAME

pcontext:SetShapeRenderer – set shape renderer

SYNOPSIS

```
pcontext:SetShapeRenderer([func[, userdata]])
```

FUNCTION

Sets callback function for context to use for rendering attributes of type `#PANGO_ATTR_SHAPE`.

The callback function will receive three to four arguments: The first argument will be a handle to a Cairo context, the second argument will be a Pango attribute of type shape and the third argument will be a boolean that indicates whether only the shape path should be appended to current path of the Cairo context and no filling/stroking should be done. If you specify the `userdata` parameter, it will be forwarded to the callback function as the fourth parameter.

To disable shape rendering, call this function without any arguments.

Note that you must make sure that the object you call this function on stays valid for as long as you need the shape renderer so make sure that you don't set it to `Nil` and make sure that the object isn't garbage-collected in any way. See the ShapeText.hws example that comes with Pangomonium for an example on how to make sure that the object you call this function on doesn't get garbage collected.

INPUTS

`func`
: optional: callback function for rendering attributes of type `#PANGO_ATTR_SHAPE`

`userdata`
: optional: user data that will be passed to `func`

19.33 pcontext:UpdateContext

NAME

pcontext:UpdateContext – update context

SYNOPSIS

```
pcontext:UpdateContext(context)
```

FUNCTION

Updates a Pango context previously created for use with Cairo to match the current transformation and target surface of the Cairo context passed in `context`.

If any layouts have been created for the context, it's necessary to call playout:ContextChanged() on those layouts.

INPUTS

`context` a Cairo context

20 Pango coverage

20.1 pcoverage:Copy

NAME
pcoverage:Copy – copy coverage

SYNOPSIS
```
handle = pcoverage:Copy()
```

FUNCTION
Copy an existing Pango coverage.

This function returns the newly allocated Pango coverage, with a reference count of one, which should be freed with pcoverage:Free().

INPUTS
none

RESULTS
`handle` the newly allocated Pango coverage,

20.2 pcoverage:Free

NAME
pcoverage:Free – free coverage

SYNOPSIS
```
pcoverage:Free()
```

FUNCTION
Decrease the reference count on the Pango coverage by one. If the result is zero, free the coverage and all associated memory.

INPUTS
none

20.3 pcoverage:Get

NAME
pcoverage:Get – get coverage level

SYNOPSIS
```
level = pcoverage:Get(index)
```

FUNCTION
Determine whether a particular index is covered by the coverage. This function returns the coverage level of the coverage for character `index`.

INPUTS
`index` the index to check

RESULTS

`level` the coverage level for character `index`

20.4 pcoverage:IsNull

NAME

pcoverage:IsNull – check if coverage is invalid

SYNOPSIS

```
bool = pcoverage:IsNull()
```

FUNCTION

Returns `True` if the coverage is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the coverage will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the coverage is `NULL`, otherwise `False`

20.5 pcoverage:Reference

NAME

pcoverage:Reference – increase reference count

SYNOPSIS

```
pcoverage:Reference()
```

FUNCTION

Increase the reference count on the Pango coverage by one.

INPUTS

none

20.6 pcoverage:Set

NAME

pcoverage:Set – set coverage level

SYNOPSIS

```
pcoverage:Set(index, level)
```

FUNCTION

Modify a particular index within the coverage.

INPUTS

`index`	the index to modify
`level`	the new level for `index`

21 Pango font

21.1 pfont:Describe

NAME
pfont:Describe – get font description

SYNOPSIS
```
desc = pfont:Describe()
```

FUNCTION
Returns a description of the font, with font size set in points.

Use pfont:DescribeWithAbsoluteSize() if you want the font size in device units.

This function returns a newly-allocated Pango font description object. You are responsible for freeing this object using pfontdesc:Free().

INPUTS
none

RESULTS
`desc` a newly-allocated Pango font description object

21.2 pfont:DescribeWithAbsoluteSize

NAME
pfont:DescribeWithAbsoluteSize – describe with absolute size

SYNOPSIS
```
desc = pfont:DescribeWithAbsoluteSize()
```

FUNCTION
Returns a description of the font, with absolute font size set in device units.

Use pfont:Describe() if you want the font size in points.

This function returns a newly-allocated Pango font description object. You are responsible for freeing this object using pfontdesc:Free().

INPUTS
none

RESULTS
`desc` a newly-allocated Pango font description object

21.3 pfont:Free

NAME
pfont:Free – free font

SYNOPSIS

```
pfont:Free()
```

FUNCTION

Decrease the reference count of a Pango font by one.

If the result is zero, the font and all associated memory will be freed.

INPUTS

none

21.4 pfont:GetCoverage

NAME

pfont:GetCoverage – get coverage map

SYNOPSIS

```
cov = pfont:GetCoverage(language)
```

FUNCTION

Computes the coverage map for a given font and language tag. The `language` parameter must be set to a Pango language object.

This function returns a newly-allocated Pango coverage object. You are responsible for freeing this object using pcoverage:Free().

INPUTS

`language`	the language tag

RESULTS

`cov`	a newly-allocated Pango coverage

21.5 pfont:GetFontMap

NAME

pfont:GetFontMap – get font map

SYNOPSIS

```
fontmap = pfont:GetFontMap()
```

FUNCTION

Gets the font map for which the font was created.

Note that the font maintains a *weak* reference to the font map, so if all references to font map are dropped, the font map will be finalized even if there are fonts created with the font map that are still alive. In that case this function will return a `NULL` object.

It is the responsibility of the user to ensure that the font map is kept alive. In most uses this is not an issue as a Pango context holds a reference to the font map.

You mustn't free the returned font map.

INPUTS

none

RESULTS

`fontmap` the Pango font map

21.6 pfont:GetGlyphExtents

NAME

pfont:GetGlyphExtents – get glyph extents

SYNOPSIS

```
ink_rect, logical_rect = pfont:GetGlyphExtents(glyph)
```

FUNCTION

Gets the logical and ink extents of a glyph within a font. This function returns two tables that have the fields the `x`, `y`, `width`, and `height` initialized.

The coordinate system for each rectangle has its origin at the base line and horizontal origin of the character with increasing coordinates extending to the right and down. The macros PANGO_ASCENT(), PANGO_DESCENT(), PANGO_LBEARING(), and PANGO_RBEARING() can be used to convert from the extents rectangle to more traditional font metrics. The units of the rectangles are in 1/#PANGO_SCALE of a device unit.

If the font is a `NULL` object, this function gracefully sets some sane values in the output variables and returns.

INPUTS

`glyph` the glyph index

RESULTS

`ink_rect` table containing the extents of the glyph as drawn

`logical_rect`
table containing the logical extents of the glyph

21.7 pfont:GetMetrics

NAME

pfont:GetMetrics – get font metrics

SYNOPSIS

```
metrics = pfont:GetMetrics([language])
```

FUNCTION

Gets overall metric information for a font.

Since the metrics may be substantially different for different scripts, a language tag can be provided to indicate that the metrics should be retrieved that correspond to the script(s) used by that language. If it is specified, the `language` parameter must be a Pango language object.

If the font is a `NULL` object, this function gracefully sets some sane values in the output variables and returns.

This function returns a Pango font metrics object. The caller must call pfontmetrics:Free() when finished using the object.

INPUTS

`language` optional: language tag used to determine which script to get the metrics for; if this parameter is omitted metrics for the entire font will be returned

RESULTS

`metrics` a Pango font metrics object

21.8 pfont:GetScaledFont

NAME

pfont:GetScaledFont – get scaled font

SYNOPSIS

```
handle = pfont:GetScaledFont()
```

FUNCTION

Gets the Cairo scaled font object used by the Pango font. The scaled font can be referenced and kept using cscaledfont:Reference().

INPUTS

none

RESULTS

`handle` a Cairo scaled font object

21.9 pfont:HasChar

NAME

pfont:HasChar – check if font has glyph

SYNOPSIS

```
ok = pfont:HasChar(wc)
```

FUNCTION

Returns whether the font provides a glyph for this character.

INPUTS

`wc` a Unicode character

RESULTS

`ok` `True` if the font can render the glyph, `False` otherwise

21.10 pfont:IsNull

NAME
pfont:IsNull – check if font is invalid

SYNOPSIS
```
bool = pfont:IsNull()
```

FUNCTION
Returns `True` if the font is `NULL`, i.e. invalid. If functions that allocate fonts fail, they might not throw an error but simply set the font to `NULL`. You can use this function to check if font allocation has failed in which case the font will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS
none

RESULTS

`bool`	`True` if the font is `NULL`, otherwise `False`

21.11 pfont:Reference

NAME
pfont:Reference – increase reference count

SYNOPSIS
```
pfont:Reference()
```

FUNCTION
Increase the reference count on the Pango font by one.

INPUTS
none

22 Pango font description

22.1 pfontdesc:BetterMatch

NAME

pfontdesc:BetterMatch – match font descriptions

SYNOPSIS

```
ok = pfontdesc:BetterMatch(old_match, new_match)
```

FUNCTION

Determines if the style attributes of `new_match` are a closer match for the font description than those of `old_match` are, or if `old_match` is `Nil`, determines if `new_match` is a match at all.

Approximate matching is done for weight and style; other style attributes must match exactly. Style attributes are all attributes other than family and size-related attributes. Approximate matching for style considers `#PANGO_STYLE_OBLIQUE` and `#PANGO_STYLE_ITALIC` as matches, but not as good a match as when the styles are equal.

Note that `old_match` must match the font description.

INPUTS

`old_match`
: a Pango font description object or `Nil`

`new_match`
: a Pango font description object

RESULTS

`ok` `True` if `new_match` is a better match

22.2 pfontdesc:Copy

NAME

pfontdesc:Copy – copy font description

SYNOPSIS

```
desc = pfontdesc:Copy()
```

FUNCTION

Make a copy of a Pango font description.

This function returns the newly allocated Pango font description, which should be freed with pfontdesc:Free().

INPUTS

none

RESULTS

`desc` the newly allocated Pango font description

22.3 pfontdesc:Equal

NAME
pfontdesc:Equal – compare font descriptions for equality

SYNOPSIS
```
ok = pfontdesc:Equal(desc2)
```

FUNCTION
Compares two font descriptions for equality.

Two font descriptions are considered equal if the fonts they describe are provably identical. This means that their masks do not have to match, as long as other fields are all the same. (Two font descriptions may result in identical fonts being loaded, but still compare `False`.)

This function returns `True` if the two font descriptions are identical, `False` otherwise.

INPUTS

`desc2`	another Pango font description object

RESULTS

`ok`	`True` if the two font descriptions are identical, otherwise `False`

22.4 pfontdesc:Free

NAME
pfontdesc:Free – free font description

SYNOPSIS
```
pfontdesc:Free()
```

FUNCTION
Frees a font description.

INPUTS
none

22.5 pfontdesc:GetFamily

NAME
pfontdesc:GetFamily – get family name

SYNOPSIS
```
family$ = pfontdesc:GetFamily()
```

FUNCTION
Gets the family name field of a font description. See pfontdesc:SetFamily() for more information.

This function returns the family name field for the font description, or `Nil` if not previously set.

INPUTS
none

RESULTS
`family$` the family name

22.6 pfontdesc:GetGravity

NAME
pfontdesc:GetGravity – get font gravity

SYNOPSIS
```
gravity = pfontdesc:GetGravity()
```

FUNCTION
Gets the gravity field of a font description. See pfontdesc:SetGravity() for more information.

This function returns the gravity field for the font description. Use pfontdesc:GetSetFields() to find out if the field was explicitly set or not.

INPUTS
none

RESULTS
`gravity` the gravity field for the font description

22.7 pfontdesc:GetSetFields

NAME
pfontdesc:GetSetFields – determine which fields have been set

SYNOPSIS
```
mask = pfontdesc:GetSetFields()
```

FUNCTION
Determines which fields in a font description have been set.

This function returns a bitmask with bits set corresponding to the fields in the font description that have been set. The following bits can be set:

`#PANGO_FONT_MASK_FAMILY`
: The font family is specified.

`#PANGO_FONT_MASK_STYLE`
: The font style is specified.

`#PANGO_FONT_MASK_VARIANT`
: The font variant is specified.

`#PANGO_FONT_MASK_WEIGHT`
: The font weight is specified.

`#PANGO_FONT_MASK_STRETCH`
: The font stretch is specified.

`#PANGO_FONT_MASK_SIZE`
: The font size is specified.

`#PANGO_FONT_MASK_GRAVITY`
: The font gravity is specified.

`#PANGO_FONT_MASK_VARIATIONS`
: OpenType font variations are specified.

INPUTS

none

RESULTS

`mask` a bitmask with bits set corresponding to the set fields (see above)

22.8 pfontdesc:GetSize

NAME

pfontdesc:GetSize – get size

SYNOPSIS

```
size = pfontdesc:GetSize()
```

FUNCTION

Gets the size field of a font description. See pfontdesc:SetSize() for more information.

This function returns the size field for the font description in points or device units. You must call pfontdesc:GetSizeIsAbsolute() to find out which is the case. Returns 0 if the size field has not previously been set or it has been set to 0 explicitly. Use pfontdesc:GetSetFields() to find out if the field was explicitly set or not.

INPUTS

none

RESULTS

`size` the size field for the font description in points

22.9 pfontdesc:GetSizeIsAbsolute

NAME

pfontdesc:GetSizeIsAbsolute – determine whether the size is in device units

SYNOPSIS

```
isabs = pfontdesc:GetSizeIsAbsolute()
```

FUNCTION

Determines whether the size of the font is in points (not absolute) or device units (absolute).

See pfontdesc:SetSize() and pfontdesc:SetAbsoluteSize() for more information.

This function returns whether the size for the font description is in points or device units. Use pfontdesc:GetSetFields() to find out if the size field of the font description was explicitly set or not.

INPUTS

none

RESULTS

`isabs` whether the size for the font description is in absolute units

22.10 pfontdesc:GetStretch

NAME

pfontdesc:GetStretch – get stretch field

SYNOPSIS

```
stretch = pfontdesc:GetStretch()
```

FUNCTION

Gets the stretch field of a font description. See pfontdesc:SetStretch() for more information.

This function returns the stretch field for the font description. Use pfontdesc:GetSetFields() to find out if the field was explicitly set or not.

INPUTS

none

RESULTS

`stretch` the stretch field for the font description

22.11 pfontdesc:GetStyle

NAME

pfontdesc:GetStyle – get style field

SYNOPSIS

```
style = pfontdesc:GetStyle()
```

FUNCTION

Gets the style field of a Pango font description. See pfontdesc:SetStyle() for more information.

This function returns the style field for the font description. Use pfontdesc:GetSetFields() to find out if the field was explicitly set or not.

INPUTS

none

RESULTS

`style` the style field for the font description

22.12 pfontdesc:GetVariant

NAME
pfontdesc:GetVariant – get variant field

SYNOPSIS
```
var = pfontdesc:GetVariant()
```

FUNCTION
Gets the variant field of a Pango font description. See pfontdesc:SetVariant() for more information.

This function returns the variant field for the font description. Use pfontdesc:GetSetFields() to find out if the field was explicitly set or not.

INPUTS
none

RESULTS

`var`	the variant field for the font description

22.13 pfontdesc:GetVariations

NAME
pfontdesc:GetVariations – get variations field

SYNOPSIS
```
var$ = pfontdesc:GetVariations()
```

FUNCTION
Gets the variations field of a font description. See pfontdesc:SetVariations() for more information.

This function returns the variations field for the font description, or `Nil` if not previously set.

INPUTS
none

RESULTS

`var$`	the variations field for the font description

22.14 pfontdesc:GetWeight

NAME
pfontdesc:GetWeight – get weight field

SYNOPSIS
```
weight = pfontdesc:GetWeight()
```

FUNCTION

Gets the weight field of a font description. See pfontdesc:SetWeight() for more information.

This function returns the weight field for the font description. Use pfontdesc:GetSetFields() to find out if the field was explicitly set or not.

INPUTS

none

RESULTS

`weight` the weight field for the font description

22.15 pfontdesc:IsNull

NAME

pfontdesc:IsNull – check if font description is invalid

SYNOPSIS

```
bool = pfontdesc:IsNull()
```

FUNCTION

Returns `True` if the font description is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the font description will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the font description is `NULL`, otherwise `False`

22.16 pfontdesc:Merge

NAME

pfontdesc:Merge – merge fields

SYNOPSIS

```
pfontdesc:Merge(desc_to_merge, replace_existing)
```

FUNCTION

Merges the fields that are set in `desc_to_merge` into the fields in the font description.

If `replace_existing` is `False`, only fields in the font description that are not already set are affected. If `True`, then fields that are already set will be replaced as well.

INPUTS

`desc_to_merge`
the Pango font description object to merge from

`replace_existing`
: if `True`, replace fields in the font description with the corresponding values from `desc_to_merge`, even if they are already exist.

22.17 pfontdesc:SetAbsoluteSize

NAME

pfontdesc:SetAbsoluteSize – set absolute size

SYNOPSIS

```
pfontdesc:SetAbsoluteSize(size)
```

FUNCTION

Sets the size field of a font description, in device units.

This is mutually exclusive with pfontdesc:SetSize() which sets the font size in points.

INPUTS

`size` the new size, in Pango units; there are `#PANGO_SCALE` Pango units in one device unit; for an output backend where a device unit is a pixel, a size value of 10 * `#PANGO_SCALE` gives a 10 pixel font

22.18 pfontdesc:SetFamily

NAME

pfontdesc:SetFamily – set family

SYNOPSIS

```
pfontdesc:SetFamily(family$)
```

FUNCTION

Sets the family name field of a font description.

The family name represents a family of related font styles, and will resolve to a particular Pango font family. In some uses of Pango font description, it is also possible to use a comma separated list of family names for this field.

INPUTS

`family$` a string representing the family name

22.19 pfontdesc:SetGravity

NAME

pfontdesc:SetGravity – set gravity

SYNOPSIS

```
pfontdesc:SetGravity(gravity)
```

FUNCTION

Sets the gravity field of a font description.

The gravity field specifies how the glyphs should be rotated. If `gravity` is `#PANGO_GRAVITY_AUTO`, this actually unsets the gravity mask on the font description.

This function is seldom useful to the user. Gravity should normally be set on a Pango context.

See pcontext:SetBaseGravity() for a list of constants that can be passed in the `gravity` argument.

INPUTS

`gravity` the gravity for the font description

22.20 pfontdesc:SetSize

NAME

pfontdesc:SetSize – set size

SYNOPSIS

```
pfontdesc:SetSize(size)
```

FUNCTION

Sets the size field of a font description in fractional points.

This is mutually exclusive with pfontdesc:SetAbsoluteSize().

INPUTS

`size` the size of the font in points, scaled by `#PANGO_SCALE` (that is, a size value of 10 * `#PANGO_SCALE` is a 10 point font; the conversion factor between points and device units depends on system configuration and the output device; for screen display, a logical DPI of 96 is common, in which case a 10 point font corresponds to a 10 * (96 / 72) = 13.3 pixel font. Usepfontdesc:SetAbsoluteSize() if you need a particular size in device units

22.21 pfontdesc:SetStretch

NAME

pfontdesc:SetStretch – set stretch field

SYNOPSIS

```
pfontdesc:SetStretch(stretch)
```

FUNCTION

Sets the stretch field of a font description. The stretch field specifies how narrow or wide the font should be.

The following constants can be passed in the `stretch` parameter:

`#PANGO_STRETCH_ULTRA_CONDENSED`
: Ultra condensed width.

`#PANGO_STRETCH_EXTRA_CONDENSED`
: Extra condensed width.

`#PANGO_STRETCH_CONDENSED`
: Condensed width.

`#PANGO_STRETCH_SEMI_CONDENSED`
: Semi condensed width.

`#PANGO_STRETCH_NORMAL`
: The normal width.

`#PANGO_STRETCH_SEMI_EXPANDED`
: Semi expanded width.

`#PANGO_STRETCH_EXPANDED`
: Expanded width.

`#PANGO_STRETCH_EXTRA_EXPANDED`
: Extra expanded width.

`#PANGO_STRETCH_ULTRA_EXPANDED`
: Ultra expanded width.

INPUTS

`stretch` the stretch for the font description (see above)

22.22 pfontdesc:SetStyle

NAME

pfontdesc:SetStyle – set style field

SYNOPSIS

```
pfontdesc:SetStyle(style)
```

FUNCTION

Sets the style field of a Pango font description.

The style field describes whether the font is slanted and the manner in which it is slanted; it can be either `#PANGO_STYLE_NORMAL`, `#PANGO_STYLE_ITALIC`, or `#PANGO_STYLE_OBLIQUE`.

Most fonts will either have a italic style or an oblique style, but not both, and font matching in Pango will match italic specifications with oblique fonts and vice-versa if an exact match is not found.

INPUTS

`style` the style for the font description (see above)

22.23 pfontdesc:SetVariant

NAME
pfontdesc:SetVariant – set variant field

SYNOPSIS
`pfontdesc:SetVariant(variant)`

FUNCTION
Sets the variant field of a font description.

The variant field can either be `#PANGO_VARIANT_NORMAL` or `#PANGO_VARIANT_SMALL_CAPS`.

INPUTS

`variant` the variant type for the font description (see above)

22.24 pfontdesc:SetVariations

NAME
pfontdesc:SetVariations – set variations

SYNOPSIS
`pfontdesc:SetVariations(variations$)`

FUNCTION
Sets the variations field of a font description.

OpenType font variations allow to select a font instance by specifying values for a number of axes, such as width or weight.

The format of the variations string is

```
AXIS1=VALUE,AXIS2=VALUE...
```

with each AXIS a 4 character tag that identifies a font axis, and each VALUE a floating point number. Unknown axes are ignored, and values are clamped to their allowed range.

Pango does not currently have a way to find supported axes of a font. Both harfbuzz and freetype have API for this.

INPUTS

`variations$`
a string representing the variations

22.25 pfontdesc:SetWeight

NAME
pfontdesc:SetWeight – set weight field

SYNOPSIS
`pfontdesc:SetWeight(weight)`

FUNCTION

Sets the weight field of a font description.

The weight field specifies how bold or light the font should be. This can be a numeric value between 100 and 1000 or one of the following predefined weight constants:

`#PANGO_WEIGHT_THIN`
: The thin weight (= 100).

`#PANGO_WEIGHT_ULTRALIGHT`
: The ultralight weight (= 200).

`#PANGO_WEIGHT_LIGHT`
: The light weight (= 300).

`#PANGO_WEIGHT_SEMILIGHT`
: The semilight weight (= 350).

`#PANGO_WEIGHT_BOOK`
: The book weight (= 380).

`#PANGO_WEIGHT_NORMAL`
: The default weight (= 400).

`#PANGO_WEIGHT_MEDIUM`
: The medium weight (= 500).

`#PANGO_WEIGHT_SEMIBOLD`
: The semibold weight (= 600).

`#PANGO_WEIGHT_BOLD`
: The bold weight (= 700).

`#PANGO_WEIGHT_ULTRABOLD`
: The ultrabold weight (= 800).

`#PANGO_WEIGHT_HEAVY`
: The heavy weight (= 900).

`#PANGO_WEIGHT_ULTRAHEAVY`
: The ultraheavy weight (= 1000).

INPUTS

`weight` the weight for the font description (see above)

22.26 pfontdesc:ToFilename

NAME

pfontdesc:ToFilename – create filename of font description

SYNOPSIS

```
f$ = pfontdesc:ToFilename()
```

FUNCTION

Creates a filename representation of a font description.

The filename is identical to the result from calling pfontdesc:ToString(), but with underscores instead of characters that are untypical in filenames, and in lower case only.

INPUTS

none

RESULTS

`f$` filename representation of font description

22.27 pfontdesc:ToString

NAME

pfontdesc:ToString – create string of font description

SYNOPSIS

```
s$ = pfontdesc:ToString()
```

FUNCTION

Creates a string representation of a font description.

See pango.FontDescription() for a description of the format of the string representation. The family list in the string description will only have a terminating comma if the last word of the list is a valid style option.

INPUTS

none

RESULTS

`s$` string representation of font description

22.28 pfontdesc:UnsetFields

NAME

pfontdesc:UnsetFields – unset fields

SYNOPSIS

```
pfontdesc:UnsetFields(to_unset)
```

FUNCTION

Unsets some of the fields in a Pango font description. The unset fields will get back to their default values.

See pfontdesc:GetSetFields() for a list of supported fields.

INPUTS

`to_unset` bitmask of fields in the the font description to unset

23 Pango font face

23.1 pfontface:Describe

NAME
pfontface:Describe – return font description

SYNOPSIS
```
desc = pfontface:Describe()
```

FUNCTION
Returns a font description that matches the face.

The resulting font description will have the family, style, variant, weight and stretch of the face, but its size field will be unset.

This function returns a newly-created Pango font description structure holding the description of the face. Use pfontdesc:Free() to free the result.

INPUTS
none

RESULTS

`desc`	a newly-created Pango font description structure

23.2 pfontface:GetFaceName

NAME
pfontface:GetFaceName – get face name

SYNOPSIS
```
name$ = pfontface:GetFaceName()
```

FUNCTION
Gets a name representing the style of this face.

Note that a font family may contain multiple faces with the same name (e.g. a variable and a non-variable face for the same style).

INPUTS
none

RESULTS

`name$`	the face name for the face

23.3 pfontface:IsNull

NAME
pfontface:IsNull – check if font face is invalid

SYNOPSIS

```
bool = pfontface:IsNull()
```

FUNCTION

Returns `True` if the font face is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the font face will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool`	`True` if the font face is `NULL`, otherwise `False`

23.4 pfontface:IsSynthesized

NAME

pfontface:IsSynthesized – check if font face is synthesized

SYNOPSIS

```
ok = pfontface:IsSynthesized()
```

FUNCTION

Returns whether a Pango font face is synthesized.

This will be the case if the underlying font rendering engine creates this face from another face, by shearing, emboldening, lightening or modifying it in some other way.

INPUTS

none

RESULTS

`ok`	boolean indicating whether the font face is synthesized

23.5 pfontface:ListSizes

NAME

pfontface:ListSizes – list sizes

SYNOPSIS

```
t = pfontface:ListSizes()
```

FUNCTION

List the available sizes for a font. This returns a table containing a list of sizes for the font. The sizes returned are in Pango units and are sorted in ascending order.

This is only applicable to bitmap fonts. For scalable fonts, an empty table is returned.

INPUTS

none

RESULTS

`t` table containing a list of font sizes

24 Pango font family

24.1 pfontfamily:GetName

NAME
pfontfamily:GetName – get family name

SYNOPSIS
```
name$ = pfontfamily:GetName()
```

FUNCTION
Gets the name of the family.

The name is unique among all fonts for the font backend and can be used in a Pango font description to specify that a face from this family is desired.

INPUTS
none

RESULTS

`name$`	the name of the family

24.2 pfontfamily:IsMonospace

NAME
pfontfamily:IsMonospace – check if family is monospace

SYNOPSIS
```
ok = pfontfamily:IsMonospace()
```

FUNCTION
A monospace font is a font designed for text display where the the characters form a regular grid.

For Western languages this would mean that the advance width of all characters are the same, but this categorization also includes Asian fonts which include double-width characters: characters that occupy two grid cells.

The best way to find out the grid-cell size is to call pfontmetrics:GetApproximateDigitWidth(), since the results of pfontmetrics:GetApproximateCharWidth() may be affected by double-width characters.

INPUTS
none

RESULTS

`ok`	`True` if the family is monospace, `False` otherwise

24.3 pfontfamily:IsNull

NAME
pfontfamily:IsNull – check if font family is invalid

SYNOPSIS
```
bool = pfontfamily:IsNull()
```

FUNCTION
Returns `True` if the font family is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the font family will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS
none

RESULTS

`bool`	`True` if the font family is `NULL`, otherwise `False`

24.4 pfontfamily:IsVariable

NAME
pfontfamily:IsVariable – check if family is variable

SYNOPSIS
```
ok = pfontfamily:IsVariable()
```

FUNCTION
A variable font is a font which has axes that can be modified to produce different faces.

Such axes are also known as variations; see pfontdesc:SetVariations() for more information.

INPUTS
none

RESULTS

`ok`	`True` if the family is variable, `False` otherwise

24.5 pfontfamily:ListFaces

NAME
pfontfamily:ListFaces – list faces inside the family

SYNOPSIS
```
t = pfontfamily:ListFaces()
```

FUNCTION

Lists the different font faces that make up the font family. The faces in a family share a common design, but differ in slant, weight, width and other aspects.

This function will return a table containing all family faces as Pango font face objects. You mustn't free the individual Pango font face objects because they are owned by the family. Note that the returned faces are not in any particular order, and multiple faces may have the same name or characteristics.

INPUTS

none

RESULTS

`t`	table containing a list of Pango font face objects

25 Pango font map

25.1 pfontmap:Changed

NAME
pfontmap:Changed – force context change

SYNOPSIS
```
pfontmap:Changed()
```

FUNCTION
Forces a change in the context, which will cause any Pango context using this fontmap to change.

This function is only useful when implementing a new backend for Pango, something applications won't do. Backends should call this function if they have attached extra data to the context and such data is changed.

INPUTS
none

25.2 pfontmap:CreateContext

NAME
pfontmap:CreateContext – create context

SYNOPSIS
```
context = pfontmap:CreateContext()
```

FUNCTION
Creates a Pango context connected to the fontmap.

This is equivalent to pango.Context() followed by pcontext:SetFontMap().

This function returns the newly allocated Pango context, which should be freed with pcontext:Free().

INPUTS
none

RESULTS
`context` the newly allocated Pango context

25.3 pfontmap:Free

NAME
pfontmap:Free – free fontmap

SYNOPSIS
```
pfontmap:Free()
```

FUNCTION
Decrease the reference count of a Pango fontmap by one.

If the result is zero, the fontmap and all associated memory will be freed.

INPUTS
none

25.4 pfontmap:GetFontType

NAME
pfontmap:GetFontType – get font type

SYNOPSIS
```
type = pfontmap:GetFontType()
```

FUNCTION
Gets the type of Cairo font backend that the fontmap uses. See cfontface:GetType() for a list of Cairo font types.

INPUTS
none

RESULTS
`type` the Cairo font backend type (see above)

25.5 pfontmap:GetResolution

NAME
pfontmap:GetResolution – get resolution

SYNOPSIS
```
dpi = pfontmap:GetResolution()
```

FUNCTION
Gets the resolution for the fontmap.

See pfontmap:SetResolution() for more information.

INPUTS
none

RESULTS
`dpi` the resolution in "dots per inch"

25.6 pfontmap:GetSerial

NAME
pfontmap:GetSerial – get serial

SYNOPSIS

```
s = pfontmap:GetSerial()
```

FUNCTION

Returns the current serial number of the fontmap.

The serial number is initialized to an small number larger than zero when a new fontmap is created and is increased whenever the fontmap is changed. It may wrap, but will never have the value 0. Since it can wrap, never compare it with "less than", always use "not equals".

The fontmap can only be changed using backend-specific API, like changing fontmap resolution.

This can be used to automatically detect changes to a Pango font map, like in Pango context.

This function returns the current serial number of the fontmap.

INPUTS

none

RESULTS

`s`	the current serial number of the fontmap

25.7 pfontmap:IsNull

NAME

pfontmap:IsNull – check if fontmap is invalid

SYNOPSIS

```
bool = pfontmap:IsNull()
```

FUNCTION

Returns `True` if the fontmap is `NULL`, i.e. invalid. If functions that allocate fontmaps fail, they might not throw an error but simply set the fontmap to `NULL`. You can use this function to check if fontmap allocation has failed in which case the fontmap will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool`	`True` if the fontmap is `NULL`, otherwise `False`

25.8 pfontmap:ListFamilies

NAME

pfontmap:ListFamilies – list families

SYNOPSIS

```
t = pfontmap:ListFamilies()
```

FUNCTION

List all families for a fontmap. The families are returned as a table containing a list of Pango font family handles in arbitrary order. The font family handles are owned by the fontmap and mustn't be freed.

INPUTS

none

RESULTS

`t`	table containing all font families

25.9 pfontmap:LoadFont

NAME

pfontmap:LoadFont – load font

SYNOPSIS

```
font = pfontmap:LoadFont(context, desc)
```

FUNCTION

Load the font in the fontmap that is the closest match for `desc`.

This function returns the newly allocated Pango font loaded, or a `NULL` object if no font matched. You can use pfont:IsNull() to check if a font is `NULL`.

INPUTS

`context`	the Pango context the font will be used with
`desc`	a Pango font description describing the font to load

RESULTS

`font`	the newly allocated Pango font

25.10 pfontmap:LoadFontset

NAME

pfontmap:LoadFontset – load fontset

SYNOPSIS

```
pfontmap:LoadFontset(context, desc, language)
```

FUNCTION

Load a set of fonts in the fontmap that can be used to render a font matching `desc`.

This function returns the newly allocated 'PangoFontset' loaded, or `Nil` if no font matched.

INPUTS

`context`	the Pango context the font will be used with

`desc` a Pango font description describing the font to load

`language` a Pango language the fonts will be used for

RESULTS

`x` the newly allocated

25.11 pfontmap:Reference

NAME

pfontmap:Reference – increase reference count

SYNOPSIS

```
pfontmap:Reference()
```

FUNCTION

Increase the reference count on the Pango fontmap by one.

INPUTS

none

25.12 pfontmap:SetResolution

NAME

pfontmap:SetResolution – set resolution

SYNOPSIS

```
pfontmap:SetResolution(dpi)
```

FUNCTION

Sets the resolution for the fontmap.

This is a scale factor between points specified in a Pango font description and Cairo units. The default value is 96, meaning that a 10 point font will be 13 units high. (10 * 96. / 72. = 13.3).

INPUTS

`dpi` the resolution in "dots per inch"; physical inches aren't actually involved; the terminology is conventional

26 Pango font metrics

26.1 pfontmetrics:Free

NAME
pfontmetrics:Free – free font metrics

SYNOPSIS
```
pfontmetrics:Free()
```

FUNCTION
Decrease the reference count of a font metrics object by one.

If the result is zero, frees the object and any associated memory.

INPUTS
none

26.2 pfontmetrics:GetApproximateCharWidth

NAME
pfontmetrics:GetApproximateCharWidth – get approximate char width

SYNOPSIS
```
width = pfontmetrics:GetApproximateCharWidth()
```

FUNCTION
Gets the approximate character width for a font metrics structure.

This is merely a representative value useful, for example, for determining the initial size for a window. Actual characters in text will be wider and narrower than this.

INPUTS
none

RESULTS
`width` the character width, in Pango units

26.3 pfontmetrics:GetApproximateDigitWidth

NAME
pfontmetrics:GetApproximateDigitWidth – get approximate digit width

SYNOPSIS
```
width = pfontmetrics:GetApproximateDigitWidth()
```

FUNCTION
Gets the approximate digit width for a font metrics structure.

This is merely a representative value useful, for example, for determining the initial size for a window. Actual digits in text can be wider or narrower than this, though this value is generally somewhat more accurate than the result of pfontmetrics:GetApproximateCharWidth() for digits.

INPUTS
none

RESULTS
`width` the digit width, in Pango units

26.4 pfontmetrics:GetAscent

NAME
pfontmetrics:GetAscent – get ascent

SYNOPSIS
```
ascent = pfontmetrics:GetAscent()
```

FUNCTION
Gets the ascent from a font metrics structure.

The ascent is the distance from the baseline to the logical top of a line of text. The logical top may be above or below the top of the actual drawn ink. It is necessary to lay out the text to figure where the ink will be.

INPUTS
none

RESULTS
`ascent` the ascent, in Pango units

26.5 pfontmetrics:GetDescent

NAME
pfontmetrics:GetDescent – get descent

SYNOPSIS
```
descent = pfontmetrics:GetDescent()
```

FUNCTION
Gets the descent from a font metrics structure.

The descent is the distance from the baseline to the logical bottom of a line of text. The logical bottom may be above or below the bottom of the actual drawn ink. It is necessary to lay out the text to figure where the ink will be.

INPUTS
none

RESULTS
`descent` the descent, in Pango units

26.6 pfontmetrics:GetHeight

NAME
pfontmetrics:GetHeight – get height

SYNOPSIS
```
height = pfontmetrics:GetHeight()
```

FUNCTION
Gets the line height from a font metrics structure.

The line height is the recommended distance between successive baselines in wrapped text using this font.

If the line height is not available, 0 is returned.

INPUTS
none

RESULTS
`height` the height, in Pango units

26.7 pfontmetrics:GetStrikethroughPosition

NAME
pfontmetrics:GetStrikethroughPosition – get strikethrough position

SYNOPSIS
```
pos = pfontmetrics:GetStrikethroughPosition()
```

FUNCTION
Gets the suggested position to draw the strikethrough.

The value returned is the distance *above* the baseline of the top of the strikethrough.

INPUTS
none

RESULTS
`pos` the suggested strikethrough position, in Pango units

26.8 pfontmetrics:GetStrikethroughThickness

NAME
pfontmetrics:GetStrikethroughThickness – get strikethrough thickness

SYNOPSIS
```
thick = pfontmetrics:GetStrikethroughThickness()
```

FUNCTION
Gets the suggested thickness to draw for the strikethrough.

INPUTS
none

RESULTS

`thick` the suggested strikethrough thickness, in Pango units

26.9 pfontmetrics:GetUnderlinePosition

NAME

pfontmetrics:GetUnderlinePosition – get underline position

SYNOPSIS

```
pos = pfontmetrics:GetUnderlinePosition()
```

FUNCTION

Gets the suggested position to draw the underline.

The value returned is the distance *above* the baseline of the top of the underline. Since most fonts have underline positions beneath the baseline, this value is typically negative.

INPUTS

none

RESULTS

`pos` the suggested underline position, in Pango units

26.10 pfontmetrics:GetUnderlineThickness

NAME

pfontmetrics:GetUnderlineThickness – get underline thickness

SYNOPSIS

```
thick = pfontmetrics:GetUnderlineThickness()
```

FUNCTION

Gets the suggested thickness to draw for the underline.

INPUTS

none

RESULTS

`thick` the suggested underline thickness, in Pango units

26.11 pfontmetrics:IsNull

NAME

pfontmetrics:IsNull – check if font metrics object is invalid

SYNOPSIS

```
bool = pfontmetrics:IsNull()
```

FUNCTION

Returns `True` if the object is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the object will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the object is `NULL`, otherwise `False`

26.12 pfontmetrics:Reference

NAME

pfontmetrics:Reference – increase reference count

SYNOPSIS

```
pfontmetrics:Reference()
```

FUNCTION

Increase the reference count of a font metrics structure by one.

INPUTS

none

27 Pango font set

27.1 pfontset:ForEach

NAME
pfontset:ForEach – iterate through all fonts

SYNOPSIS
`pfontset:ForEach(func[, userdata])`

FUNCTION
Iterates through all the fonts in a fontset, calling `func` for each one. If `func` returns `True`, that stops the iteration.

The callback function will receive the fontset in the first parameter and the Pango font in the second parameter. If you specify the `userdata` parameter, it will be passed to the callback as the third parameter.

INPUTS

`func`	a callback function
`data`	additional data to pass to the callback function

27.2 pfontset:Free

NAME
pfontset:Free – free fontset

SYNOPSIS
`pfontset:Free()`

FUNCTION
Frees a fontset.

INPUTS
none

27.3 pfontset:GetFont

NAME
pfontset:GetFont – get font

SYNOPSIS
`font = pfontset:GetFont(wc)`

FUNCTION
Returns the font in the fontset that contains the best glyph for a Unicode character. You must free the font returned by this function using pfont:Free().

INPUTS

`wc`	a Unicode character

RESULTS

`font` a Pango font object

27.4 pfontset:GetMetrics

NAME

pfontset:GetMetrics – get metrics for all fonts

SYNOPSIS

```
metrics = pfontset:GetMetrics()
```

FUNCTION

Get overall metric information for the fonts in the fontset. You must free the font metrics object returned by this function using pfontmetrics:Free().

INPUTS

none

RESULTS

`metrics` a Pango font metrics object

27.5 pfontset:IsNull

NAME

pfontset:IsNull – check if font set is invalid

SYNOPSIS

```
bool = pfontset:IsNull()
```

FUNCTION

Returns `True` if the fontset is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the fontset will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the fontset is `NULL`, otherwise `False`

27.6 pfontset:Reference

NAME

pfontset:Reference – increase reference count

SYNOPSIS

```
pfontset:Reference()
```

FUNCTION

Increase the reference count on the Pango fontset by one.

INPUTS

none

28 Pango glyph item

28.1 pglyphitem:Copy

NAME
pglyphitem:Copy – copy glyph item

SYNOPSIS
```
handle = pglyphitem:Copy()
```

FUNCTION
Make a deep copy of an existing Pango glyph item structure.

INPUTS
none

RESULTS
`handle` the newly allocated Pango glyph item

28.2 pglyphitem:Free

NAME
pglyphitem:Free – free glyph item

SYNOPSIS
```
pglyphitem:Free()
```

FUNCTION
Frees a Pango glyph item and resources to which it points.

INPUTS
none

28.3 pglyphitem:Get

NAME
pglyphitem:Get – get glyph item data

SYNOPSIS
```
t = pglyphitem:Get()
```

FUNCTION
Gets information about the glyph item. The data is returned as a table. See pglyphitem:Set() for a description of all the fields that will be set in the table.

INPUTS
none

RESULTS
`t` table containing information about the glyph item (see above)

28.4 pglyphitem:GetItem

NAME
pglyphitem:GetItem – get glyph item

SYNOPSIS
```
item = pglyphitem:GetItem()
```

FUNCTION
Gets the Pango item associated with the glyph item. You mustn't free this object.

INPUTS
none

RESULTS
`item` a Pango item object

28.5 pglyphitem:GetGlyphString

NAME
pglyphitem:GetGlyphString – get glyph string

SYNOPSIS
```
glyph_string = pglyphitem:GetGlyphString()
```

FUNCTION
Gets the glyph string associated with the glyph item. You mustn't free this object.

INPUTS
none

RESULTS
`glyph_string`
a Pango glyph string object

28.6 pglyphitem:GetLogicalWidths

NAME
pglyphitem:GetLogicalWidths – get logical widths

SYNOPSIS
```
logical_widths = pglyphitem:GetLogicalWidths(text$)
```

FUNCTION
Given a Pango glyph item and the corresponding text, determine the width corresponding to each character. The individual widths are returned in a table. When multiple characters compose a single cluster, the width of the entire cluster is divided equally among the characters.

See also pglyphstring:GetLogicalWidths().

INPUTS

`text$` text that the glyph item corresponds to

RESULTS

`logical_widths`
table containing the resulting character widths

28.7 pglyphitem:IsNull

NAME

pglyphitem:IsNull – check if glyph item is invalid

SYNOPSIS

```
bool = pglyphitem:IsNull()
```

FUNCTION

Returns `True` if the glyph item is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the glyph item will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the glyph item is `NULL`, otherwise `False`

28.8 pglyphitem:Set

NAME

pglyphitem:Set – set glyph item data

SYNOPSIS

```
pglyphitem:Set(table)
```

FUNCTION

Sets data of one or more data fields inside the glyph item. You have to pass a table to this function that consists of one or more key-value pairs, each setting a single data member.

The following keys are recognized in the `table` parameter:

`YOffset` Shift of the baseline, relative to the baseline of the containing line. Positive values shift upwards.

`StartXOffset`
Horizontal displacement to apply before the glyph item. Positive values shift right.

`EndXOffset`
: Horizontal displacement to apply after th glyph item. Positive values shift right.

INPUTS

`table` table containing key-value pairs with the data to set (see above)

28.9 pglyphitem:SetItem

NAME

pglyphitem:SetItem – set glyph item

SYNOPSIS

```
pglyphitem:SetItem(item)
```

FUNCTION

Sets the Pango item inside the glyph item to `item`. Note that `item` won't be referenced by this function so you need to make sure it stays valid for the glyph item's lifecycle.

INPUTS

`item` a Pango item object

28.10 pglyphitem:SetGlyphString

NAME

pglyphitem:SetGlyphString – set glyph string

SYNOPSIS

```
pglyphitem:SetGlyphString(glyph_string)
```

FUNCTION

Sets the glyph string inside the glyph item to `glyph_string`. Note that `glyph_string` won't be referenced by this function so you need to make sure it stays valid for the glyph item's lifecycle.

INPUTS

`glyph_string`
: a Pango glyph string object

28.11 pglyphitem:Split

NAME

pglyphitem:Split – split item

SYNOPSIS

```
handle = pglyphitem:Split(text$, split_index)
```

FUNCTION

Modifies the glyph item to cover only the text after `split_index`, and returns a new item that covers the text before `split_index` that used to be in the glyph item.

You can think of `split_index` as the length of the returned item. `split_index` may not be 0, and it may not be greater than or equal to the length of the source glyph item (that is, there must be at least one byte assigned to each item, you can't create a zero-length item).

This function returns the newly allocated item representing text before `split_index`, which should be freed with pglyphitem:Free().

INPUTS

`text$` text to which positions in the glyph item apply

`split_index`
byte index of position to split item, relative to the start of the item

RESULTS

`handle` the newly allocated glyph item

29 Pango glyph string

29.1 pglyphstring:Copy

NAME

pglyphstring:Copy – copy glyph string

SYNOPSIS

```
gstr = pglyphstring:Copy()
```

FUNCTION

Copy a glyph string and associated storage.

INPUTS

none

RESULTS

`gstr` the newly allocated Pango glyph string

29.2 pglyphstring:Extents

NAME

pglyphstring:Extents – compute glyph string extents

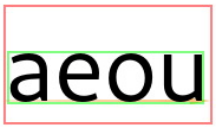
SYNOPSIS

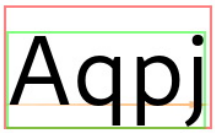
```
ink_rect, logical_rect = pglyphstring:Extents(font)
```

FUNCTION

Compute the logical and ink extents of a glyph string. This will return two tables that have the `x`, `y`, `width`, and `height` fields initialized. See the documentation for pfont:GetGlyphExtents() for details about the interpretation of the rectangles.

Examples of logical (red) and ink (green) rects:





INPUTS

`font` a Pango font

RESULTS

`ink_rect` table containing the extents of the glyph string as drawn

`logical_rect`
table containing the logical extents of the glyph string

29.3 pglyphstring:ExtentsRange

NAME
pglyphstring:ExtentsRange – extents range

SYNOPSIS
```
ink_rect, logical_rect = pglyphstring:ExtentsRange(start, end, font)
```

FUNCTION
Computes the extents of a sub-portion of a glyph string. This will return two tables that have the `x`, `y`, `width`, and `height` fields initialized.

The extents are relative to the start of the glyph string range (the origin of their coordinate system is at the start of the range, not at the start of the entire glyph string).

INPUTS

`start` start index

`end` end index (the range is the set of bytes with indices such that start <= index < end)

`font` a Pango font

RESULTS

`ink_rect` table containing the extents of the glyph string range as drawn

`logical_rect`
table containing the logical extents of the glyph string range

29.4 pglyphstring:Free

NAME
pglyphstring:Free – free glyph string

SYNOPSIS
```
pglyphstring:Free()
```

FUNCTION
Free a glyph string and associated storage.

INPUTS
none

29.5 pglyphstring:Get

NAME
pglyphstring:Get – get glyph info

SYNOPSIS
```
table = pglyphstring:Get(index)
```

FUNCTION

Gets positioning information and visual attributes for the glyph at `index`. The information is returned as a table. See pglyphstring:Set() for a description of all the table fields that are set by this function.

INPUTS

`index` glyph to use

RESULTS

`table` a table containing glyph information

29.6 pglyphstring:GetLogicalWidths

NAME

pglyphstring:GetLogicalWidths – get logical widths

SYNOPSIS

```
t = pglyphstring:GetLogicalWidths(text$, embedding_level)
```

FUNCTION

Given a Pango glyph string and corresponding text, determine the width corresponding to each character. When multiple characters compose a single cluster, the width of the entire cluster is divided equally among the characters. The individual widths are returned in a table.

See also pglyphitem:GetLogicalWidths().

INPUTS

`text$` the text corresponding to the glyphs

`embedding_level`
the embedding level of the string

RESULTS

`logical_widths`
table containing the resulting character widths

29.7 pglyphstring:GetWidth

NAME

pglyphstring:GetWidth – get width

SYNOPSIS

```
width = pglyphstring:GetWidth()
```

FUNCTION

Computes the logical width of the glyph string.

This can also be computed using pglyphstring:Extents(). However, since this only computes the width, it's much faster. This is in fact only a convenience function that computes the sum of `geometry.width` for each glyph in the glyph string.

INPUTS
none

RESULTS

`width` the logical width of the glyph string

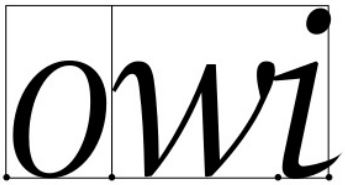
29.8 pglyphstring:IndexToX

NAME
pglyphstring:IndexToX – convert from character to x position

SYNOPSIS
```
x_pos = pglyphstring:IndexToX(text$, analysis, index, trailing)
```

FUNCTION
Converts from character position to x position.

The X position is measured from the left edge of the run. Character positions are obtained using font metrics for ligatures where available, and computed by dividing up each cluster into equal portions, otherwise.



INPUTS

`text$` the text for the run

`analysis` the analysis information return from itemize; this must be a Pango analysis object

`index` the byte index within `text$`

`trailing` whether we should compute the result for the beginning (`False`) or end (`True`) of the character

RESULTS

`x_pos` location to store result

29.9 pglyphstring:IsNull

NAME

pglyphstring:IsNull – check if glyph string is invalid

SYNOPSIS

```
bool = pglyphstring:IsNull()
```

FUNCTION

Returns `True` if the glyph string is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the glyph string will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool` `True` if the glyph string is `NULL`, otherwise `False`

29.10 pglyphstring:Set

NAME

pglyphstring:Set – set glyph info

SYNOPSIS

```
pglyphstring:Set(index, table)
```

FUNCTION

Sets positioning information and visual attributes for the glyph at `index`. The data to set must be passed as one or more key-value pairs in the `table` argument. The following keys are currently supported:

`Glyph` The glyph itself.

`Width` The logical width to use for the the character.

`XOffset` Horizontal offset from nominal character position.

`YOffset` Vertical offset from nominal character position.

`IsClusterStart`
Set for the first logical glyph in each cluster.

`LogCluster`
Logical cluster info, indexed by the byte index within the text corresponding to the glyph string.

INPUTS

`index` glyph to use

`table` a table consisting of key-value pairs of the data to set

29.11 pglyphstring:SetSize

NAME
pglyphstring:SetSize – resize glyph string

SYNOPSIS
`pglyphstring:SetSize(new_len)`

FUNCTION
Resize a glyph string to the given length.

INPUTS

`new_len` the new length of the string

29.12 pglyphstring:XToIndex

NAME
pglyphstring:XToIndex – convert from x offset to character position

SYNOPSIS
`index, trailing = pglyphstring:XToIndex(text$, analysis, x_pos)`

FUNCTION
Convert from x offset to character position.

Character positions are computed by dividing up each cluster into equal portions. In scripts where positioning within a cluster is not allowed (such as Thai), the returned value may not be a valid cursor position; the caller must combine the result with the logical attributes for the text to compute the valid cursor position.

INPUTS

`text$` the text for the run

`analysis` the analysis information return from itemize; this must be a Pango analysis object

`x_pos` the x offset (in Pango units)

RESULTS

`index` calculated byte index within `text$`

`trailing` boolean indicating whether the user clicked on the leading or trailing edge of the character

30 Pango item

30.1 pitem:Copy

NAME
pitem:Copy – copy item

SYNOPSIS
`item = pitem:Copy()`

FUNCTION
Copy an existing Pango item structure.

INPUTS
none

RESULTS
`item` the newly allocated Pango item

30.2 pitem:Free

NAME
pitem:Free – free item

SYNOPSIS
`pitem:Free()`

FUNCTION
Free a Pango item and all associated memory.

INPUTS
none

30.3 pitem:Get

NAME
pitem:Get – get item data

SYNOPSIS
`table = pitem:Get()`

FUNCTION
Gets data for the item. The data will be returned as a table. See pitem:Set() for information on which table fields will be set.

INPUTS
none

RESULTS
`table` a table containing the item data

30.4 pitem:IsNull

NAME
pitem:IsNull – check if item is invalid

SYNOPSIS
`bool = pitem:IsNull()`

FUNCTION
Returns `True` if the item is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the item will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS
none

RESULTS

`bool`	`True` if the item is `NULL`, otherwise `False`

30.5 pitem:Set

NAME
pitem:Set – set item data

SYNOPSIS
`pitem:Set(table)`

FUNCTION
Sets data for the item. The data to set must be passed as one or more key-value pairs in the `table` argument. The following keys are currently supported:

`Offset`	Byte offset of the start of this item in text.
`Length`	Length of this item in bytes.
`NumChars`	Number of Unicode characters in the item.
`Analysis`	Analysis results for the item. This is a Pango analysis object.

INPUTS

`table`	a table consisting of key-value pairs of the data to set

30.6 pitem:Split

NAME
pitem:Split – split item

SYNOPSIS
`item = pitem:Split(split_index, split_offset)`

FUNCTION

Modifies the item to cover only the text after `split_index`, and returns a new item that covers the text before `split_index` that used to be in the original item.

You can think of `split_index` as the length of the returned item. `split_index` may not be 0, and it may not be greater than or equal to the length of the item (that is, there must be at least one byte assigned to each item, you can't create a zero-length item). `split_offset` is the length of the first item in chars, and must be provided because the text used to generate the item isn't available, so this function can't count the char length of the split items itself.

This function returns new item representing text before `split_index`, which should be freed with pitem:Free().

INPUTS

`split_index`
: byte index of position to split item, relative to the start of the item

`split_offset`
: number of chars between start of the item and `split_index`

RESULTS

`item` new item representing text before `split_index`

31 Pango language

31.1 planguage:GetSampleString

NAME

planguage:GetSampleString – get sample string

SYNOPSIS

```
s$ = planguage:GetSampleString()
```

FUNCTION

Get a string that is representative of the characters needed to render a particular language.

The sample text may be a pangram, but is not necessarily. It is chosen to be demonstrative of normal text in the language, as well as exposing font feature requirements unique to the language. It is suitable for use as sample text in a font selection dialog.

If Pango does not have a sample string for the language, the classic "The quick brown fox..." is returned. This can be detected by comparing the returned handle value to that returned for (non-existent) language code "xx".

INPUTS

none

RESULTS

`s$` sample string for the language

31.2 planguage:GetScripts

NAME

planguage:GetScripts – get language scripts

SYNOPSIS

```
t = planguage:GetScripts()
```

FUNCTION

Determines the scripts used to write the language. The individual scripts are returned in a table. The list of scripts returned starts with the script that the language uses most and continues to the one it uses least.

Most languages use only one script for writing, but there are some that use two (Latin and Cyrillic for example), and a few use three (Japanese for example).

The following scripts are currently defined:

`#PANGO_SCRIPT_INVALID_CODE`
: Signals an invalid script.

`#PANGO_SCRIPT_COMMON`
: A character used by multiple different scripts.

`#PANGO_SCRIPT_INHERITED`
: A mark glyph that takes its script from the base glyph to which it is attached.

`#PANGO_SCRIPT_ARABIC`
: Arabic.

`#PANGO_SCRIPT_ARMENIAN`
: Armenian.

`#PANGO_SCRIPT_BENGALI`
: Bengali.

`#PANGO_SCRIPT_BOPOMOFO`
: Bopomofo.

`#PANGO_SCRIPT_CHEROKEE`
: Cherokee.

`#PANGO_SCRIPT_COPTIC`
: Coptic.

`#PANGO_SCRIPT_CYRILLIC`
: Cyrillic.

`#PANGO_SCRIPT_DESERET`
: Deseret.

`#PANGO_SCRIPT_DEVANAGARI`
: Devanagari.

`#PANGO_SCRIPT_ETHIOPIC`
: Ethiopic.

`#PANGO_SCRIPT_GEORGIAN`
: Georgian.

`#PANGO_SCRIPT_GOTHIC`
: Gothic.

`#PANGO_SCRIPT_GREEK`
: Greek.

`#PANGO_SCRIPT_GUJARATI`
: Gujarati.

`#PANGO_SCRIPT_GURMUKHI`
: Gurmukhi.

`#PANGO_SCRIPT_HAN`
: Han.

`#PANGO_SCRIPT_HANGUL`
: Hangul.

`#PANGO_SCRIPT_HEBREW`
: Hebrew.

`#PANGO_SCRIPT_HIRAGANA`
: Hiragana.

`#PANGO_SCRIPT_KANNADA`
: Kannada.

`#PANGO_SCRIPT_KATAKANA`
: Katakana.

`#PANGO_SCRIPT_KHMER`
: Khmer.

`#PANGO_SCRIPT_LAO`
: Lao.

`#PANGO_SCRIPT_LATIN`
: Latin.

`#PANGO_SCRIPT_MALAYALAM`
: Malayalam.

`#PANGO_SCRIPT_MONGOLIAN`
: Mongolian.

`#PANGO_SCRIPT_MYANMAR`
: Myanmar.

`#PANGO_SCRIPT_OGHAM`
: Ogham.

`#PANGO_SCRIPT_OLD_ITALIC`
: Old Italic.

`#PANGO_SCRIPT_ORIYA`
: Oriya.

`#PANGO_SCRIPT_RUNIC`
: Runic.

`#PANGO_SCRIPT_SINHALA`
: Sinhala.

`#PANGO_SCRIPT_SYRIAC`
: Syriac.

`#PANGO_SCRIPT_TAMIL`
: Tamil.

`#PANGO_SCRIPT_TELUGU`
: Telugu.

`#PANGO_SCRIPT_THAANA`
: Thaana.

`#PANGO_SCRIPT_THAI`
: Thai.

`#PANGO_SCRIPT_TIBETAN`
: Tibetan.

`#PANGO_SCRIPT_CANADIAN_ABORIGINAL`
: Canadian Aboriginal.

`#PANGO_SCRIPT_YI`
: Yi.

`#PANGO_SCRIPT_TAGALOG`
: Tagalog.

`#PANGO_SCRIPT_HANUNOO`
: Hanunoo.

`#PANGO_SCRIPT_BUHID`
: Buhid.

`#PANGO_SCRIPT_TAGBANWA`
: Tagbanwa.

`#PANGO_SCRIPT_BRAILLE`
: Braille.

`#PANGO_SCRIPT_CYPRIOT`
: Cypriot.

`#PANGO_SCRIPT_LIMBU`
: Limbu.

`#PANGO_SCRIPT_OSMANYA`
: Osmanya.

`#PANGO_SCRIPT_SHAVIAN`
: Shavian.

`#PANGO_SCRIPT_LINEAR_B`
: Linear B.

`#PANGO_SCRIPT_TAI_LE`
: Tai Le.

`#PANGO_SCRIPT_UGARITIC`
: Ugaritic.

`#PANGO_SCRIPT_NEW_TAI_LUE`
: New Tai Lue.

`#PANGO_SCRIPT_BUGINESE`
: Buginese.

`#PANGO_SCRIPT_GLAGOLITIC`
: Glagolitic.

`#PANGO_SCRIPT_TIFINAGH`
: Tifinagh.

`#PANGO_SCRIPT_SYLOTI_NAGRI`
: Syloti Nagri.

`#PANGO_SCRIPT_OLD_PERSIAN`
: Old Persian.

`#PANGO_SCRIPT_KHAROSHTHI`
: Kharoshthi.

`#PANGO_SCRIPT_UNKNOWN`
: An unassigned code point.

`#PANGO_SCRIPT_BALINESE`
: Balinese.

`#PANGO_SCRIPT_CUNEIFORM`
: Cuneiform.

`#PANGO_SCRIPT_PHOENICIAN`
: Phoenician.

`#PANGO_SCRIPT_PHAGS_PA`
: Phags-pa.

`#PANGO_SCRIPT_NKO`
: NKo.

`#PANGO_SCRIPT_KAYAH_LI`
: Kayah Li.

`#PANGO_SCRIPT_LEPCHA`
: Lepcha.

`#PANGO_SCRIPT_REJANG`
: Rejang.

`#PANGO_SCRIPT_SUNDANESE`
: Sundanese.

`#PANGO_SCRIPT_SAURASHTRA`
: Saurashtra.

`#PANGO_SCRIPT_CHAM`
: Cham.

`#PANGO_SCRIPT_OL_CHIKI`
: Ol Chiki.

`#PANGO_SCRIPT_VAI`
: Vai.

`#PANGO_SCRIPT_CARIAN`
: Carian.

`#PANGO_SCRIPT_LYCIAN`
: Lycian.

`#PANGO_SCRIPT_LYDIAN`
: Lydian.

`#PANGO_SCRIPT_BATAK`
: Batak.

`#PANGO_SCRIPT_BRAHMI`
: Brahmi.

`#PANGO_SCRIPT_MANDAIC`
: Mandaic.

`#PANGO_SCRIPT_CHAKMA`
: Chakma.

`#PANGO_SCRIPT_MEROITIC_CURSIVE`
: Meroitic Cursive.

`#PANGO_SCRIPT_MEROITIC_HIEROGLYPHS`
: Meroitic Hieroglyphs.

`#PANGO_SCRIPT_MIAO`
: Miao.

`#PANGO_SCRIPT_SHARADA`
: Sharada.

`#PANGO_SCRIPT_SORA_SOMPENG`
: Sora Sompeng.

`#PANGO_SCRIPT_TAKRI`
: Takri.

`#PANGO_SCRIPT_BASSA_VAH`
: Bassa.

`#PANGO_SCRIPT_CAUCASIAN_ALBANIAN`
: Caucasian Albanian.

`#PANGO_SCRIPT_DUPLOYAN`
: Duployan.

`#PANGO_SCRIPT_ELBASAN`
: Elbasan.

`#PANGO_SCRIPT_GRANTHA`
: Grantha.

`#PANGO_SCRIPT_KHOJKI`
: Kjohki.

`#PANGO_SCRIPT_KHUDAWADI`
: Khudawadi, Sindhi.

`#PANGO_SCRIPT_LINEAR_A`
: Linear A.

`#PANGO_SCRIPT_MAHAJANI`
: Mahajani.

`#PANGO_SCRIPT_MANICHAEAN`
: Manichaean.

`#PANGO_SCRIPT_MENDE_KIKAKUI`
: Mende Kikakui.

`#PANGO_SCRIPT_MODI`
: Modi.

`#PANGO_SCRIPT_MRO`
: Mro.

`#PANGO_SCRIPT_NABATAEAN`
: Nabataean.

`#PANGO_SCRIPT_OLD_NORTH_ARABIAN`
: Old North Arabian.

`#PANGO_SCRIPT_OLD_PERMIC`
: Old Permic.

`#PANGO_SCRIPT_PAHAWH_HMONG`
: Pahawh Hmong.

`#PANGO_SCRIPT_PALMYRENE`
: Palmyrene.

`#PANGO_SCRIPT_PAU_CIN_HAU`
: Pau Cin Hau.

`#PANGO_SCRIPT_PSALTER_PAHLAVI`
: Psalter Pahlavi.

`#PANGO_SCRIPT_SIDDHAM`
: Siddham.

`#PANGO_SCRIPT_TIRHUTA`
: Tirhuta.

`#PANGO_SCRIPT_WARANG_CITI`
: Warang Citi.

`#PANGO_SCRIPT_AHOM`
: Ahom.

`#PANGO_SCRIPT_ANATOLIAN_HIEROGLYPHS`
: Anatolian Hieroglyphs.

`#PANGO_SCRIPT_HATRAN`
: Hatran.

`#PANGO_SCRIPT_MULTANI`
: Multani.

`#PANGO_SCRIPT_OLD_HUNGARIAN`
: Old Hungarian.

`PANGO_SCRIPT_SIGNWRITING`
: Signwriting.

INPUTS

none

RESULTS

`t` table containing the individual scripts (see above)

31.3 planguage:IncludesScript

NAME

planguage:IncludesScript – check if language includes script

SYNOPSIS

```
ok = planguage:IncludesScript(script)
```

FUNCTION

Determines if `script` is one of the scripts used to write the language.

The returned value is conservative; if nothing is known about the language tag the language, `True` will be returned, since, as far as Pango knows, `script` might be used to write the language.

This routine is used in Pango's itemization process when determining if a supplied language tag is relevant to a particular section of text. It probably is not useful for applications in most circumstances.

See planguage:GetScripts() for a list of supported scripts.

INPUTS

`script` a script identifier

RESULTS

`ok` `True` if `script` is one of the scripts used to write the language, `False` otherwise

31.4 planguage:IsNull

NAME

planguage:IsNull – check if language is invalid

SYNOPSIS

```
bool = planguage:IsNull()
```

FUNCTION

Returns `True` if the language is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the language will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS
none

RESULTS
`bool` `True` if the language is `NULL`, otherwise `False`

31.5 planguage:Matches

NAME
planguage:Matches – check if language tag matches language ranges

SYNOPSIS
```
ok = planguage:Matches(range_list$)
```

FUNCTION
Checks if a language tag matches one of the elements in a list of language ranges. The list of language ranges must be passed as a string, separated by ";", ":", ",", or space characters. Each element must either be "*", or a RFC 3066 language range canonicalized as by pango.Language().

A language tag is considered to match a range in the list if the range is "*", the range is exactly the tag, or the range is a prefix of the tag, and the character after it in the tag is "-".

This function returns `True` if a match was found

INPUTS
`range_list$` a list of language ranges (see above for the format)

RESULTS
`ok` `True` if a match was found, `False` otherwise

31.6 planguage:ToString

NAME
planguage:ToString – convert language tag to RFC-3066

SYNOPSIS
```
f$ = planguage:ToString()
```

FUNCTION
Gets the RFC-3066 format string representing the given language tag.

INPUTS
none

RESULTS
`f$` a string representing the language tag

32 Pango layout

32.1 playout:ContextChanged

NAME
playout:ContextChanged – force recomputation of layout

SYNOPSIS
```
playout:ContextChanged()
```

FUNCTION
Forces recomputation of any state in the Pango layout that might depend on the layout's context.

This function should be called if you make changes to the context subsequent to creating the layout.

INPUTS
none

32.2 playout:Copy

NAME
playout:Copy – copy layout

SYNOPSIS
```
handle = playout:Copy()
```

FUNCTION
Creates a deep copy-by-value of the layout.

The attribute list, tab array, and text from the original layout are all copied by value.

INPUTS
none

RESULTS
`handle` the newly allocated Pango layout

32.3 playout:Free

NAME
playout:Free – free layout

SYNOPSIS
```
playout:Free()
```

FUNCTION
Decrease the reference count of a Pango layout by one.

If the result is zero, the layout and all associated memory will be freed.

INPUTS
none

32.4 playout:GetAlignment

NAME
playout:GetAlignment – get layout alignment

SYNOPSIS
```
align = playout:GetAlignment()
```

FUNCTION
Gets the alignment for the layout: how partial lines are positioned within the horizontal space available. See playout:SetAlignment() for a list of alignments.

INPUTS
none

RESULTS
`align` the alignment

32.5 playout:GetAttributes

NAME
playout:GetAttributes – get attribute list

SYNOPSIS
```
attrlist = playout:GetAttributes()
```

FUNCTION
Gets the attribute list for the layout, if any. If there is no attribute list in the layout, a `NULL` object is returned. You can use pattrlist:IsNull() to check if an attribute list is `NULL`.

INPUTS
none

RESULTS
`attrlist` a Pango attribute list

32.6 playout:GetAutoDir

NAME
playout:GetAutoDir – get layout's auto dir setting

SYNOPSIS
```
autodir = playout:GetAutoDir()
```

FUNCTION

Gets whether to calculate the base direction for the layout according to its contents.

See playout:SetAutoDir() for more information.

This function returns `True` if the bidirectional base direction is computed from the layout's contents, `False` otherwise.

INPUTS

none

RESULTS

`autodir` `True` if auto dir is on, `False` otherwise

32.7 playout:GetBaseline

NAME

playout:GetBaseline – get baseline

SYNOPSIS

```
baseline = playout:GetBaseline()
```

FUNCTION

Gets the Y position of baseline of the first line in the layout.

INPUTS

none

RESULTS

`baseline` baseline of first line, from top of the layout

32.8 playout:GetCharacterCount

NAME

playout:GetCharacterCount – get character count

SYNOPSIS

```
n = playout:GetCharacterCount()
```

FUNCTION

Returns the number of Unicode characters in the the text of the layout.

INPUTS

none

RESULTS

`n` the number of Unicode characters

32.9 playout:GetContext

NAME
playout:GetContext – get Pango context

SYNOPSIS
```
ctx = playout:GetContext()
```

FUNCTION
Retrieves the Pango context used for this layout. The context is owned by the layout and mustn't be freed.

INPUTS
none

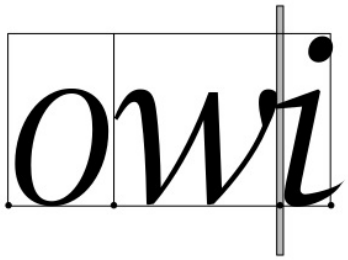
RESULTS
`ctx` the Pango context for the layout

32.10 playout:GetCursorPos

NAME
playout:GetCursorPos – get cursor pos

SYNOPSIS
```
strong_pos, weak_pos = playout:GetCursorPos(index)
```

FUNCTION
Given an `index` within a layout, determines the positions that of the strong and weak cursors if the insertion point is at that index. Both `strongpos` and `weakpos` will be tables that have the fields `x`, `y`, `width`, and `height` initialized. The position of each cursor is stored as a zero-width rectangle with the height of the run extents.



The strong cursor location is the location where characters of the directionality equal to the base direction of the layout are inserted. The weak cursor location is the location where characters of the directionality opposite to the base direction of the layout are inserted.

The following example shows text with both a strong and a weak cursor.

The strong cursor has a little arrow pointing to the right, the weak cursor to the left. Typing a 'c' in this situation will insert the character after the 'b', and typing another Hebrew character, like '', will insert it at the end.

INPUTS

`index` the byte index of the cursor

RESULTS

`strong_pos`
strong cursor position

`weak_pos` weak cursor position

32.11 playout:GetEllipsize

NAME

playout:GetEllipsize – get ellipsization type

SYNOPSIS

```
mode = playout:GetEllipsize()
```

FUNCTION

Gets the type of ellipsization being performed for the layout. See playout:SetEllipsize() for more information.

Use playout:IsEllipsized() to query whether any paragraphs were actually ellipsized.

INPUTS

none

RESULTS

`mode` the current ellipsization mode for the layout

32.12 playout:GetExtents

NAME

playout:GetExtents – get layout extents

SYNOPSIS

```
ink_rect, logical_rect = playout:GetExtents()
```

FUNCTION

Computes the logical and ink extents of the layout.

Logical extents are usually what you want for positioning things. Note that both extents may have non-zero x and y. You may want to use those to offset where you render the

layout. Not doing that is a very typical bug that shows up as right-to-left layouts not being correctly positioned in a layout with a set width.

The extents are given in layout coordinates and in Pango units; layout coordinates begin at the top left corner of the layout.

This function returns tables that have the fields `x`, `y`, `width`, and `height` initialized.

INPUTS

none

RESULTS

`ink_rect` extents of the layout as drawn

`logical_rect`
logical extents of the layout

32.13 playout:GetFontDescription

NAME

playout:GetFontDescription – get font description

SYNOPSIS

```
fontdesc = playout:GetFontDescription()
```

FUNCTION

Gets the font description for the layout, if any. This function returns a handle to the layout's font description, or a `NULL` object if the font description from the layout's context is inherited. You can use pfontdesc:IsNull() to check if an object is `NULL`.

INPUTS

none

RESULTS

`fontdesc` a handle to a font description

32.14 playout:GetHeight

NAME

playout:GetHeight – get layout height

SYNOPSIS

```
height = playout:GetHeight()
```

FUNCTION

Gets the height of layout used for ellipsization. See playout:SetHeight() for details.

This function returns the height, in Pango units if positive, or number of lines if negative.

INPUTS

none

RESULTS

`height` layout height

32.15 playout:GetIndent

NAME
playout:GetIndent – get paragraph indent

SYNOPSIS
```
indent = playout:GetIndent()
```

FUNCTION
Gets the paragraph indent width in Pango units. A negative value indicates a hanging indentation.

INPUTS
none

RESULTS

`indent`	the indent in Pango units

32.16 playout:GetIter

NAME
playout:GetIter – get layout iterator

SYNOPSIS
```
it = playout:GetIter()
```

FUNCTION
Returns an iterator to iterate over the visual extents of the layout. The caller is responsible for freeing the iterator using playoutiter:Free().

INPUTS
none

RESULTS

`it`	the new Pango layout iterator

32.17 playout:GetJustify

NAME
playout:GetJustify – get line justify

SYNOPSIS
```
justify = playout:GetJustify()
```

FUNCTION
Gets whether each complete line should be stretched to fill the entire width of the layout.

INPUTS
none

RESULTS

`justify`	a boolean value

32.18 playout:GetLine

NAME
playout:GetLine – get line

SYNOPSIS
```
handle = playout:GetLine(line)
```

FUNCTION
Retrieves a particular line from a Pango layout.

Use the faster playout:GetLineReadonly() if you do not plan to modify the contents of the line (glyphs, glyph widths, etc.).

This function returns the requested Pango layout line, or a `NULL` object if the index is out of range. The returned Pango layout line is owned by the layout and mustn't be freed. It can be ref'ed and retained, but will become invalid if changes are made to the Pango layout.

INPUTS

`line`	the index of a line, which must be between 0 and the number of lines in the layout minus one

RESULTS

`handle`	a Pango layout line object

32.19 playout:GetLineCount

NAME
playout:GetLineCount – get number of lines in layout

SYNOPSIS
```
n = playout:GetLineCount()
```

FUNCTION
Retrieves the count of lines for the layout.

INPUTS
none

RESULTS

`n`	the line count

32.20 playout:GetLineReadonly

NAME
playout:GetLineReadonly – get layout line in read-only mode

SYNOPSIS
```
handle = playout:GetLineReadonly(line)
```

FUNCTION

Retrieves a particular line from a Pango layout.

This is a faster alternative to playout:GetLine(), but the user is not expected to modify the contents of the line (glyphs, glyph widths, etc.).

This function returns the requested Pango layout line, or a `NULL` object if the index is out of range. The returned Pango layout line is owned by the layout and mustn't be freed. It can be ref'ed and retained, but will become invalid if changes are made to the Pango layout.

INPUTS

`line`	the index of a line, which must be between 0 and the number of lines in the layout minus one

RESULTS

`handle`	a Pango layout line object

32.21 playout:GetLineSpacing

NAME

playout:GetLineSpacing – get line spacing

SYNOPSIS

```
spacing = playout:GetLineSpacing()
```

FUNCTION

Gets the line spacing factor of the layout. See playout:SetLineSpacing() for more information.

INPUTS

none

RESULTS

`spacing`	line spacing factor

32.22 playout:GetLines

NAME

playout:GetLines – get layout lines

SYNOPSIS

```
list = playout:GetLines()
```

FUNCTION

Returns the lines of the layout as a list.

Use the faster playout:GetLinesReadonly() if you do not plan to modify the contents of the lines (glyphs, glyph widths, etc.).

This function returns a list containing the lines in the layout. The Pango layout line objects in the list must be used with care. They will become invalid on any change to the layout's text or properties.

INPUTS
none

RESULTS
`list` table containing a list of Pango layout line objects

32.23 playout:GetLinesReadonly

NAME
playout:GetLinesReadonly – get layout lines in read-only mode

SYNOPSIS
```
list = playout:GetLinesReadonly()
```

FUNCTION
Returns the lines of the layout as a list.

This is a faster alternative to playout:GetLines(), but the user is not expected to modify the contents of the lines (glyphs, glyph widths, etc.).

This function returns a list containing the lines in the layout. The Pango layout line objects in the list must be used with care. They will become invalid on any change to the layout's text or properties.

INPUTS
none

RESULTS
`list` table containing a list of Pango layout line objects

32.24 playout:GetLogAttrs

NAME
playout:GetLogAttrs – get logical attributes for characters

SYNOPSIS
```
t = playout:GetLogAttrs()
```

FUNCTION
Retrieves an array of logical attributes for each character in the layout. This function will return a table that contains a subtable for each character in the layout. Each subtable will be have the following fields initialized:

`IsLineBreak`
: If set, can break line in front of character.

`IsMandatoryBreak`
: If set, must break line in front of character.

`IsCharBreak`
: If set, can break here when doing character wrapping.

`IsWhite` Is whitespace character.

`IsCursorPosition`
: If set, cursor can appear in front of character. i.e. this is a grapheme boundary, or the first character in the text. This flag implements Unicode's Grapheme Cluster Boundaries semantics.

`IsWordStart`
: Is first character in a word.

`IsWordEnd`
: Is first non-word char after a word. Note that in degenerate cases, you could have both `IsWordStart` and `IsWordEnd` set for some character.

`IsSentenceBoundary`
: Is a sentence boundary. There are two ways to divide sentences. The first assigns all inter-sentence whitespace/control/format chars to some sentence, so all chars are in some sentence; `IsSentenceBoundary` denotes the boundaries there. The second way doesn't assign between-sentence spaces, etc. to any sentence, so `IsSentenceStart` / `IsSentenceEnd` mark the boundaries of those sentences.

`IsSentenceStart`
: Is first character in a sentence.

`IsSentenceEnd`
: Is first char after a sentence. Note that in degenerate cases, you could have both `IsSentenceStart` and `IsSentenceEnd` set for some character. (e.g. no space after a period, so the next sentence starts right away).

`BackspaceDeletesCharacter`
: If set, backspace deletes one character rather than the entire grapheme cluster. This field is only meaningful on grapheme boundaries (where `IsCursorPosition` is set). In some languages, the full grapheme (e.g. letter + diacritics) is considered a unit, while in others, each decomposed character in the grapheme is a unit. In the default implementation of pango break, this bit is set on all grapheme boundaries except those following Latin, Cyrillic or Greek base characters.

`IsExpandableSpace`
: Is a whitespace character that can possibly be expanded for justification purposes.

`IsWordBoundary`
: Is a word boundary, as defined by UAX#29. More specifically, means that this is not a position in the middle of a word. For example, both sides of a punctuation mark are considered word boundaries. This flag is particularly useful when selecting text word-by-word. This flag implements Unicodes Word Boundaries semantics.

The number of attributes returned in the table will be one more than the total number of characters in the layout, since there need to be attributes corresponding to both the position before the first character and the position after the last character.

INPUTS
none

RESULTS
`t` table containing an array of logical attributes

32.25 playout:GetPixelExtents

NAME
playout:GetPixelExtents – get pixel extents of layout

SYNOPSIS
```
ink_rect, logical_rect = playout:GetPixelExtents()
```

FUNCTION
Computes the logical and ink extents of the layout in device units. Both return values are tables that have the `x`, `y`, `width`, and `height` fields initialized.

This function just calls playout:GetExtents() followed by two pango.ExtentsToPixels() calls, rounding `ink_rect` and `logical_rect` such that the rounded rectangles fully contain the unrounded one (that is, passes them as first argument to pango.ExtentsToPixels().

INPUTS
none

RESULTS
`ink_rect` extents of the layout as drawn

`logical_rect`
logical extents of the layout

32.26 playout:GetPixelSize

NAME
playout:GetPixelSize – get layout dimensions

SYNOPSIS
```
width, height = playout:GetPixelSize()
```

FUNCTION
Determines the logical width and height of a Pango layout in device units.

playout:GetSize() returns the width and height scaled by `#PANGO_SCALE`. This is simply a convenience function around playout:GetPixelExtents().

INPUTS
none

RESULTS
`width` logical width

`height` logical height

32.27 playout:GetSerial

NAME

playout:GetSerial – get serial

SYNOPSIS

```
s = playout:GetSerial()
```

FUNCTION

Returns the current serial number of the layout.

The serial number is initialized to a small number larger than zero when a new layout is created and is increased whenever the layout is changed using any of the setter functions, or the Pango context it uses has changed. The serial may wrap, but will never have the value 0. Since it can wrap, never compare it with "less than", always use "not equals".

This can be used to automatically detect changes to a Pango layout, and is useful for example to decide whether a layout needs redrawing. To force the serial to be increased, use playout:ContextChanged().

INPUTS

none

RESULTS

`s`	the current serial number of the layout

32.28 playout:GetSingleParagraphMode

NAME

playout:GetSingleParagraphMode – get single paragraph mode

SYNOPSIS

```
bool = playout:GetSingleParagraphMode()
```

FUNCTION

Obtains whether the layout is in single paragraph mode. See playout:SetSingleParagraphMode() for details.

This function returns `True` if the layout does not break paragraphs at paragraph separator characters, `False` otherwise.

INPUTS

none

RESULTS

`bool`	`True` if the layout does not break paragraphs, `False` otherwise

32.29 playout:GetSize

NAME

playout:GetSize – get layout dimensions

SYNOPSIS

```
width, height = playout:GetSize()
```

FUNCTION

Determines the logical width and height of a Pango layout in Pango units.

This is simply a convenience function around playout:GetExtents().

INPUTS

none

RESULTS

`width`	logical width
`height`	logical height

32.30 playout:GetSpacing

NAME

playout:GetSpacing – get line spacing

SYNOPSIS

```
spacing = playout:GetSpacing()
```

FUNCTION

Gets the amount of spacing between the lines of the layout.

INPUTS

none

RESULTS

`spacing`	the spacing in Pango units

32.31 playout:GetTabs

NAME

playout:GetTabs – get tab array

SYNOPSIS

```
handle = playout:GetTabs()
```

FUNCTION

Gets the current Pango tab array used by this layout.

If no Pango tab array has been set, then the default tabs are in use and a `NULL` object is returned. You can use ptabarray:IsNull() to check if a tab array is `NULL`. Default tabs are every 8 spaces.

The return value should be freed with ptabarray:Free().

INPUTS

none

RESULTS

`handle` a Pango tab array

32.32 playout:GetText

NAME

playout:GetText – get text

SYNOPSIS

```
s$ = playout:GetText()
```

FUNCTION

Gets the text in the layout.

INPUTS

none

RESULTS

`s$` the text in the layout

32.33 playout:GetUnknownGlyphsCount

NAME

playout:GetUnknownGlyphsCount – get unknown glyphs count

SYNOPSIS

```
n = playout:GetUnknownGlyphsCount()
```

FUNCTION

Counts the number of unknown glyphs in the layout.

This function can be used to determine if there are any fonts available to render all characters in a certain string, or when used in combination with `#PANGO_ATTR_FALLBACK`, to check if a certain font supports all the characters in the string.

INPUTS

none

RESULTS

`n` the number of unknown glyphs in the layout

32.34 playout:GetWidth

NAME

playout:GetWidth – get wrap width

SYNOPSIS

```
width = playout:GetWidth()
```

FUNCTION

Gets the width to which the lines of the Pango layout should wrap. This function returns the width in Pango units, or -1 if no width set.

INPUTS

none

RESULTS

`width` the width in Pango units, or -1 if no width set

32.35 playout:GetWrap

NAME

playout:GetWrap – get wrap mode

SYNOPSIS

```
mode = playout:GetWrap()
```

FUNCTION

Gets the wrap mode for the layout. See playout:SetWrap() for a list of supported wrap modes.

Use playout:IsWrapped() to query whether any paragraphs were actually wrapped.

INPUTS

none

RESULTS

`mode` active wrap mode

32.36 playout:IndexToLineX

NAME

playout:IndexToLineX – index to line x

SYNOPSIS

```
line, x_pos = playout:IndexToLineX(index, trailing)
```

FUNCTION

Converts from byte `index` within the layout to line and X position.

The X position is measured from the left edge of the line.

INPUTS

`index` the byte index of a grapheme within the layout

`trailing` an integer indicating the edge of the grapheme to retrieve the position of. If > 0, the trailing edge of the grapheme, if 0, the leading of the grapheme

RESULTS

`line` line index (will be between 0 and the number of layout lines minus 1)

`x_pos` position within line (`#PANGO_SCALE` units per device unit)

32.37 playout:IndexToPos

NAME

playout:IndexToPos – get onscreen position of grapheme

SYNOPSIS

```
pos = playout:IndexToPos(index)
```

FUNCTION

Converts from an index within a Pango layout to the onscreen position corresponding to the grapheme at that index.

The return value is a table containing the fields `x`, `y`, `width`, and `height` to describe a rectangle. Note that `x` is always the leading edge of the grapheme and `x+width` the trailing edge of the grapheme. If the directionality of the grapheme is right-to-left, then `width` will be negative.

INPUTS

`index` byte index within the layout

RESULTS

`pos` rectangle containing the position of the grapheme

32.38 playout:IsEllipsized

NAME

playout:IsEllipsized – check for ellipsized paragraphs

SYNOPSIS

```
bool = playout:IsEllipsized()
```

FUNCTION

Queries whether the layout had to ellipsize any paragraphs.

This returns `True` if the ellipsization mode for the layout is not `#PANGO_ELLIPSIZE_NONE`, a positive width is set on the layout, and there are paragraphs exceeding that width that have to be ellipsized.

INPUTS

none

RESULTS

`bool` `True` if any paragraphs had to be ellipsized, `False` otherwise

32.39 playout:IsNull

NAME

playout:IsNull – check if layout is invalid

SYNOPSIS

```
bool = playout:IsNull()
```

FUNCTION

Returns `True` if the layout is `NULL`, i.e. invalid. If functions that allocate layouts fail, they might not throw an error but simply set the layout to `NULL`. You can use this function to check if layout allocation has failed in which case the layout will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS

none

RESULTS

`bool`	`True` if the layout is `NULL`, otherwise `False`

32.40 playout:IsWrapped

NAME

playout:IsWrapped – check for wrapped paragraphs

SYNOPSIS

```
bool = playout:IsWrapped()
```

FUNCTION

Queries whether the layout had to wrap any paragraphs.

This returns `True` if a positive width is set on the layout, ellipsization mode of the layout is set to `#PANGO_ELLIPSIZE_NONE`, and there are paragraphs exceeding the layout width that have to be wrapped.

INPUTS

none

RESULTS

`bool`	`True` if any paragraphs had to be wrapped, `False` otherwise

32.41 playout:MoveCursorVisually

NAME

playout:MoveCursorVisually – move cursor visually

SYNOPSIS

```
nidx, ntrail = playout:MoveCursorVisually(strong, oidx, otrail, dir)
```

FUNCTION

Computes a new cursor position from an old position and a direction.

If `dir` is positive, then the new position will cause the strong or weak cursor to be displayed one position to right of where it was with the old cursor position. If `dir` is negative, it will be moved to the left.

In the presence of bidirectional text, the correspondence between logical and visual order will depend on the direction of the current run, and there may be jumps when the cursor is moved off of the end of a run.

Motion here is in cursor positions, not in characters, so a single call to this function may move the cursor over multiple characters when multiple characters combine to form a single grapheme.

INPUTS

`strong` whether the moving cursor is the strong cursor or the weak cursor; the strong cursor is the cursor corresponding to text insertion in the base direction for the layout

`oidx` the byte index of the current cursor position

`otrail` if 0, the cursor was at the leading edge of the grapheme indicated by `oidx`; if > 0, the cursor was at the trailing edge

`dir` direction to move cursor; a negative value indicates motion to the left

RESULTS

`nidx` new cursor byte index; a value of -1 indicates that the cursor has been moved off the beginning of the layout; a value of `#G_MAXINT` indicates that the cursor has been moved off the end of the layout

`ntrail` number of characters to move forward from the location returned for `nidx` to get the position where the cursor should be displayed; this allows distinguishing the position at the beginning of one line from the position at the end of the preceding line; `nind` is always on the line where the cursor should be displayed

32.42 playout:Reference

NAME

playout:Reference – increase reference count

SYNOPSIS

```
playout:Reference()
```

FUNCTION

Increase the reference count on the Pango layout by one.

INPUTS

none

32.43 playout:SetAlignment

NAME

playout:SetAlignment – set alignment

SYNOPSIS

```
playout:SetAlignment(alignment)
```

FUNCTION

Sets the alignment for the layout: how partial lines are positioned within the horizontal space available. The `alignment` parameter can be one of the following constants:

`#PANGO_ALIGN_LEFT`
: Put all available space on the right.

`#PANGO_ALIGN_CENTER`
: Center the line within the available space.

`#PANGO_ALIGN_RIGHT`
: Put all available space on the left.

The default alignment is `#PANGO_ALIGN_LEFT`.

INPUTS

`alignment`
: desired alignment (see above)

32.44 playout:SetAttributes

NAME

playout:SetAttributes – set attributes

SYNOPSIS

```
playout:SetAttributes(attrs)
```

FUNCTION

Sets the text attributes for a layout object.

This function references `attrs`, so the caller can unref its reference.

INPUTS

`attrs` a Pango attribute list object

32.45 playout:SetAutoDir

NAME

playout:SetAutoDir – set auto dir

SYNOPSIS

```
playout:SetAutoDir(auto_dir)
```

FUNCTION

Sets whether to calculate the base direction for the layout according to its contents.

When this flag is on (the default), then paragraphs in the layout that begin with strong right-to-left characters (Arabic and Hebrew principally), will have right-to-left layout, paragraphs with letters from other scripts will have left-to-right layout. Paragraphs with only neutral characters get their direction from the surrounding paragraphs.

When `False`, the choice between left-to-right and right-to-left layout is done according to the base direction of the layout's Pango context. See pcontext:SetBaseDir() for details.

When the auto-computed direction of a paragraph differs from the base direction of the context, the interpretation of `#PANGO_ALIGN_LEFT` and `#PANGO_ALIGN_RIGHT` are swapped.

INPUTS

`auto_dir` if `True`, compute the bidirectional base direction from the layout's contents

32.46 playout:SetEllipsize

NAME

playout:SetEllipsize – set ellipsize

SYNOPSIS

```
playout:SetEllipsize(ellipsize)
```

FUNCTION

Sets the type of ellipsization being performed for the layout.

Depending on the ellipsization mode `ellipsize` text is removed from the start, middle, or end of text so they fit within the width and height of layout set with playout:SetWidth() and playout:SetHeight().

If the layout contains characters such as newlines that force it to be layed out in multiple paragraphs, then whether each paragraph is ellipsized separately or the entire layout is ellipsized as a whole depends on the set height of the layout.

The following values are possible for `ellipsize`:

`#PANGO_ELLIPSIZE_NONE`
: No ellipsization.

`#PANGO_ELLIPSIZE_START`
: Omit characters at the start of the text.

`#PANGO_ELLIPSIZE_MIDDLE`
: Omit characters in the middle of the text.

`#PANGO_ELLIPSIZE_END`
: Omit characters at the end of the text.

The default value is `#PANGO_ELLIPSIZE_NONE`.

See playout:SetHeight() for details.

INPUTS

`ellipsize`
: the new ellipsization mode for the layout (see above)

32.47 playout:SetFontDescription

NAME

playout:SetFontDescription – set font description

SYNOPSIS

```
playout:SetFontDescription(desc)
```

FUNCTION

Sets the default font description for the layout.

If no font description is set on the layout, the font description from the layout's context is used.

INPUTS

`desc`	the new Pango font description to unset the current font description

32.48 playout:SetHeight

NAME

playout:SetHeight – set height

SYNOPSIS

```
playout:SetHeight(height)
```

FUNCTION

Sets the height to which the Pango layout should be ellipsized at.

There are two different behaviors, based on whether `height` is positive or negative.

If `height` is positive, it will be the maximum height of the layout. Only lines would be shown that would fit, and if there is any text omitted, an ellipsis added. At least one line is included in each paragraph regardless of how small the height value is. A value of zero will render exactly one line for the entire layout.

If `height` is negative, it will be the (negative of) maximum number of lines per paragraph. That is, the total number of lines shown may well be more than this value if the layout contains multiple paragraphs of text. The default value of -1 means that the first line of each paragraph is ellipsized. This behavior may be changed in the future to act per layout instead of per paragraph.

Height setting only has effect if a positive width is set on the layout and ellipsization mode of the layout is not `#PANGO_ELLIPSIZE_NONE`. The behavior is undefined if a height other than -1 is set and ellipsization mode is set to `#PANGO_ELLIPSIZE_NONE`, and may change in the future.

INPUTS

`height`	the desired height of the layout in Pango units if positive, or desired number of lines if negative.

32.49 playout:SetIndent

NAME

playout:SetIndent – set indent

SYNOPSIS

```
playout:SetIndent(indent)
```

FUNCTION

Sets the width in Pango units to indent each paragraph.

A negative value of `indent` will produce a hanging indentation. That is, the first line will have the full width, and subsequent lines will be indented by the absolute value of `indent`.

The indent setting is ignored if layout alignment is set to `#PANGO_ALIGN_CENTER`.

The default value is 0.

INPUTS

`indent` the amount by which to indent

32.50 playout:SetJustify

NAME

playout:SetJustify – set justify mode

SYNOPSIS

```
playout:SetJustify(justify)
```

FUNCTION

Sets whether each complete line should be stretched to fill the entire width of the layout.

Stretching is typically done by adding whitespace, but for some scripts (such as Arabic), the justification may be done in more complex ways, like extending the characters.

Note that tabs and justification conflict with each other: Justification will move content away from its tab-aligned positions.

The default value is `False`.

INPUTS

`justify` whether the lines in the layout should be justified

32.51 playout:SetLineSpacing

NAME

playout:SetLineSpacing – set line spacing

SYNOPSIS

```
playout:SetLineSpacing(factor)
```

FUNCTION

Sets a factor for line spacing.

Typical values are: 0, 1, 1.5, 2. The default values is 0.

If `factor` is non-zero, lines are placed so that

```
baseline2 = baseline1 + factor * height2
```

where `height2` is the line height of the second line (as determined by the font(s)). In this case, the spacing set with playout:SetSpacing() is ignored.

If `factor` is zero (the default), spacing is applied as before.

Note: for semantics that are closer to the CSS line-height property, use the line height attribute in a Pango attribute list.

INPUTS

`factor` the new line spacing factor

32.52 playout:SetMarkup

NAME

playout:SetMarkup – set markup text

SYNOPSIS

```
playout:SetMarkup(text$[, length])
```

FUNCTION

Sets the layout text and attribute list from marked-up text. This will replace the current text and attribute list. It is the same as playout:SetMarkupWithAccel(), but the markup text isn't scanned for accelerators.

The Pango markup language allows you to apply text formatting using XML tags. The most common tag in the Pango markup language is the `<span>` tag. It allows you to apply formatting to a range of characters. The `<span>` tag supports the following attributes:

`font`

`font_desc`

A font description string, such as "Sans Italic 12". See pango.FontDescription() for a description of the format of the string representation. Note that any other span attributes will override this description. So if you have "Sans Italic" and also a style="normal" attribute, you will get Sans normal, not italic.

`font_family`

`face` A font family name.

`font_size`

`size` Font size in 1024ths of a point, or in points (e.g. "12.5pt"), or one of the absolute sizes "xx-small", "x-small", "small", "medium", "large", "x-large", "xx-large", or a percentage (e.g. "200%"), or one of the relative sizes "smaller" or "larger". If you want to specify a absolute size, it"s usually easier to take advantage of the ability to specify a partial font description using "font"; you can use font="12.5" rather than size="12800" or size="12.5pt".

`font_style`

`style` One of "normal", "oblique", "italic".

`font_weight`

`weight` One of "ultralight", "light", "normal", "bold", "ultrabold", "heavy", or a numeric weight.

`font_variant`
`variant` One of "normal", "small-caps", "all-small-caps", "petite-caps", "all-petite-caps", "unicase", "title-caps".

`font_stretch`
`stretch` One of "ultracondensed", "extracondensed", "condensed", "semicondensed", "normal", "semiexpanded", "expanded", "extraexpanded", "ultraexpanded".

`font_features`
A comma-separated list of OpenType font feature settings, in the same syntax as accepted by CSS. E.g: font_features='dlig=1, -kern, afrc on'.

`foreground`
`fgcolor`

`color` An RGB color specification such as "#00FF00" or a color name such as "red". An RGBA color specification such as "#00FF007F" will be interpreted as specifying both a foreground color and foreground alpha.

`background`
`bgcolor` An RGB color specification such as "#00FF00" or a color name such as "red". An RGBA color specification such as "#00FF007F" will be interpreted as specifying both a background color and background alpha.

`alpha`

`fgalpha` An alpha value for the foreground color, either a plain integer between 1 and 65536 or a percentage value like "50%".

`background_alpha`
`bgalpha` An alpha value for the background color, either a plain integer between 1 and 65536 or a percentage value like "50%".

`underline`
One of "none", "single", "double", "low", "error".

`underline_color`
The color of underlines; an RGB color specification such as "#00FF00" or a color name such as "red".

`overline` One of "none" or "single".

`overline_color`
The color of overlines; an RGB color specification such as "#00FF00" or a color name such as "red".

`rise` Vertical displacement, in Pango units or in points (e.g. "5pt"). Can be negative for subscript, positive for superscript.

`baseline_shift`
Vertical displacement. In contrast to rise, baseline_shift attributes are cumulative. The value can be a length in Pango units or in points (e.g. "5pt"), or "superscript" or "subscript".

`font_scale`
: Font size change. The possible values are "superscript", "subscript" or "small-caps". This is similar to the font_size values "smaller" or "larger", but uses font metrics to find the new size.

`strikethrough`
: "true" or "false" whether to strike through the text.

`strikethrough_color`
: The color of strikethrough lines; an RGB color specification such as "#00FF00" or a color name such as "red".

`fallback`
: "true" or "false" whether to enable fallback. If disabled, then characters will only be used from the closest matching font on the system. No fallback will be done to other fonts on the system that might contain the characters in the text. Fallback is enabled by default. Most applications should not disable fallback.

`lang`
: A language code, indicating the text language.

`letter_spacing`
: Inter-letter spacing in 1024ths of a point.

`gravity`
: One of "south", "east", "north", "west", "auto".

`gravity_hint`
: One of "natural", "strong", "line".

`show`
: Specifies what special characters to show visibly. The value can be "none" or a combination of "spaces", "line-breaks" and "ignorables", combined with "|".

`insert_hyphens`
: "true" or "false" to indicate whether hyphens should be inserted when breaking lines in the middle of words.

`allow_breaks`
: "true" or "false" to indicate whether breaking lines is allowed.

`line_height`
: Overrides the line height. The value can be either a factor (< 1024) that is used to scale up the logical extents of runs or an absolute value (in 1024th of a point).

`text_transform`
: Specifies how characters are transformed during shaping. The values can be "none", "lowercase", "uppercase" or "capitalize".

`segment`
: Overrides word or sentence boundaries. The value can be "word" or "sentence", to indicate that the span should be treated as a single word or sentence. Overlapping segments will be split to allow this, and line breaks will be adjusted accordingly.

Besides the `<span>` tag, the Pango markup language also supports the following convenience tags:

`<b>`
: Bold

`<big>`	Makes font relatively larger, equivalent to `<span size="larger">`
`<i>`	Italic
`<s>`	Strikethrough
`<sub>`	Subscript
`<sup>`	Superscript
`<small>`	Makes font relatively smaller, equivalent to `<span size="smaller">`
`<tt>`	Monospace font
`<u>`	Underline

Here's an example of marked-up text:

```
<span foreground="blue" size="x-large">Blue text</span> is <i>cool</i>.
```

Note that you can also use XML features such as numeric character entities, e.g. you can use `&#169;` for the copyright character etc.

INPUTS

`text$`	marked-up text
`length`	optional: length of marked-up text in bytes (defaults to -1 which means use the length of string)

32.53 playout:SetMarkupWithAccel

NAME

playout:SetMarkupWithAccel – set markup with accel

SYNOPSIS

```
accel_char = playout:SetMarkupWithAccel(text$, length, accel_marker)
```

FUNCTION

Sets the layout text and attribute list from marked-up text.

Replaces the current text and attribute list.

If `accel_marker` is nonzero, the given character will mark the character following it as an accelerator. For example, `accel_marker` might be an ampersand or underscore. All characters marked as an accelerator will receive a `#PANGO_UNDERLINE_LOW` attribute, and the first character so marked will be returned in `accel_char`. Two `accel_marker` characters following each other produce a single literal `accel_marker` character.

INPUTS

`text$`	marked-up text
`length`	length of marked-up text in bytes (-1 for length of `text$`)
`accel_marker`	marker for accelerators in the text

RESULTS

`accel_char`	return location for first located accelerator

32.54 playout:SetSingleParagraphMode

NAME

playout:SetSingleParagraphMode – set single paragraph mode

SYNOPSIS

`playout:SetSingleParagraphMode(setting)`

FUNCTION

Sets the single paragraph mode of the layout.

If `setting` is `True`, do not treat newlines and similar characters as paragraph separators; instead, keep all text in a single paragraph, and display a glyph for paragraph separator characters. Used when you want to allow editing of newlines on a single text line.

The default value is `False`.

INPUTS

`setting` new setting

32.55 playout:SetSpacing

NAME

playout:SetSpacing – set spacing

SYNOPSIS

`playout:SetSpacing(spacing)`

FUNCTION

Sets the amount of spacing in Pango units between the lines of the layout.

When placing lines with spacing, Pango arranges things so that

```
line2.top = line1.bottom + spacing
```

The default value is 0.

Note: Since 1.44, Pango is using the line height (as determined by the font) for placing lines when the line spacing factor is set to a non-zero value with playout:SetLineSpacing(). In that case, the `spacing` set with this function is ignored.

Note: for semantics that are closer to the CSS line-height property, see the line height Pango attribute.

INPUTS

`spacing` the amount of spacing

32.56 playout:SetTabs

NAME

playout:SetTabs – set tabs

SYNOPSIS

`playout:SetTabs([tabs])`

FUNCTION

Sets the tabs to use for the layout, overriding the default tabs.

Pango layout will place content at the next tab position whenever it meets a Tab character (U+0009).

By default, tabs are every 8 spaces. If the `tabs` parameter is omitted, the default tabs are reinstated. `tabs` is copied into the layout; you must free your copy of `tabs` yourself.

Note that tabs and justification conflict with each other: Justification will move content away from its tab-aligned positions. The same is true for alignments other than `#PANGO_ALIGN_LEFT`.

INPUTS

`tabs` optional: a Pango tab array

32.57 playout:SetText

NAME

playout:SetText – set text

SYNOPSIS

```
playout:SetText(text$[, length])
```

FUNCTION

Sets the text of the layout.

This function validates `text` and renders invalid UTF-8 with a placeholder glyph.

Note that if you have used playout:SetMarkup() or playout:SetMarkupWithAccel() on the layout before, you may want to call playout:SetAttributes() to clear the attributes set on the layout from the markup as this function does not clear attributes.

INPUTS

`text$` the text

`length` optional: maximum length of `text` in bytes (defaults to -1 which means length of `text$`)

32.58 playout:SetWidth

NAME

playout:SetWidth – set wrap width

SYNOPSIS

```
playout:SetWidth(width)
```

FUNCTION

Sets the width to which the lines of the Pango layout should wrap or ellipsized.

The default value is -1 which means no wrapping or ellipsization is requested.

INPUTS

`width` the desired width in Pango units, or -1 to indicate that no wrapping or ellipsization should be performed.

32.59 playout:SetWrap

NAME

playout:SetWrap – set wrap mode

SYNOPSIS

```
playout:SetWrap(wrap)
```

FUNCTION

Sets the wrap mode. The wrap mode only has effect if a width is set on the layout with playout:SetWidth(). To turn off wrapping, set the width to -1.

The following wrap modes are currently supported:

`#PANGO_WRAP_WORD`
: Wrap lines at word boundaries.

`#PANGO_WRAP_CHAR`
: Wrap lines at character boundaries.

`#PANGO_WRAP_WORD_CHAR`
: Wrap lines at word boundaries, but fall back to character boundaries if there is not enough space for a full word.

The default value is `#PANGO_WRAP_WORD`.

INPUTS

`wrap` the wrap mode (see above)

32.60 playout:XYToIndex

NAME

playout:XYToIndex – convert fro XY to byte index

SYNOPSIS

```
inside, index, trailing = playout:XYToIndex(x, y)
```

FUNCTION

Converts from X and Y position within a layout to the byte index to the character at that logical position.

If the Y position is not inside the layout, the closest position is chosen (the position will be clamped inside the layout). If the X position is not within the layout, then the start or the end of the line is chosen as described for playoutline:XToIndex(). If either the X or Y positions were not inside the layout, then the function returns FALSE; on an exact hit, it returns `True`.

This function returns `True` if the coordinates were inside text, `False` otherwise

INPUTS

`x`	the X offset (in Pango units) from the left edge of the layout
`y`	the Y offset (in Pango units) from the top edge of the layout

RESULTS

`inside`	`True` if the coordinates were inside text, `False` otherwise
`index`	calculated byte index
`trailing`	integer indicating where in the grapheme the user clicked; it will either be zero, or the number of characters in the grapheme; 0 represents the leading edge of the grapheme

33 Pango layout iterator

33.1 playoutiter:AtLastLine

NAME
playoutiter:AtLastLine – check if iterator is on last line

SYNOPSIS
```
ok = playoutiter:AtLastLine()
```

FUNCTION
Determines whether the iterator is on the last line of the layout.

INPUTS
none

RESULTS

`ok`	`True` if the iterator is on the last line

33.2 playoutiter:Copy

NAME
playoutiter:Copy – copy layout iterator

SYNOPSIS
```
iter = playoutiter:Copy()
```

FUNCTION
Copies a Pango layout iterator.

This function returns the newly allocated Pango layout iterator

INPUTS
none

RESULTS

`iter`	the newly allocated Pango layout iterator

33.3 playoutiter:Free

NAME
playoutiter:Free – free layout iterator

SYNOPSIS
```
playoutiter:Free()
```

FUNCTION
Frees an iterator that's no longer in use.

INPUTS
none

33.4 playoutiter:GetBaseline

NAME
playoutiter:GetBaseline – get baseline

SYNOPSIS
```
baseline = playoutiter:GetBaseline()
```

FUNCTION
Gets the Y position of the current line's baseline, in layout coordinates.

Layout coordinates have the origin at the top left of the entire layout.

INPUTS
none

RESULTS
`baseline` baseline of current line

33.5 playoutiter:GetCharExtents

NAME
playoutiter:GetCharExtents – get character extents

SYNOPSIS
```
logical_rect = playoutiter:GetCharExtents()
```

FUNCTION
Gets the extents of the current character, in layout coordinates. This will return a table that has the fields `x`, `y`, `width`, and `height` initialized.

Layout coordinates have the origin at the top left of the entire layout.

Only logical extents can sensibly be obtained for characters; ink extents make sense only down to the level of clusters.

INPUTS
none

RESULTS
`logical_rect`
rectangle filled with logical extents

33.6 playoutiter:GetClusterExtents

NAME
playoutiter:GetClusterExtents – get cluster extents

SYNOPSIS
```
ink_rect, logical_rect = playoutiter:GetClusterExtents()
```

FUNCTION

Gets the extents of the current cluster, in layout coordinates. This will return two tables that have the fields `x`, `y`, `width`, and `height` initialized.

Layout coordinates have the origin at the top left of the entire layout.

INPUTS

none

RESULTS

`ink_rect` rectangle filled with ink extents

`logical_rect`
rectangle filled with logical extents

33.7 playoutiter:GetIndex

NAME

playoutiter:GetIndex – get current byte index

SYNOPSIS

```
idx = playoutiter:GetIndex()
```

FUNCTION

Gets the current byte index.

Note that iterating forward by char moves in visual order, not logical order, so indices may not be sequential. Also, the index may be equal to the length of the text in the layout, if on the `Nil` run (see playoutiter:GetRun()).

INPUTS

none

RESULTS

`idx` current byte index

33.8 playoutiter:GetLayout

NAME

playoutiter:GetLayout – get iterator layout

SYNOPSIS

```
handle = playoutiter:GetLayout()
```

FUNCTION

Gets the layout associated with a Pango layout iterator. This is owned by the iterator and mustn't be freed.

INPUTS

none

RESULTS

`handle` the layout associated with the iterator

33.9 playoutiter:GetLayoutExtents

NAME
playoutiter:GetLayoutExtents – get layout extents

SYNOPSIS
```
ink_rect, logical_rect = playoutiter:GetLayoutExtents()
```

FUNCTION
Obtains the extents of the Pango layout being iterated over. This will return two tables that have the fields `x`, `y`, `width`, and `height` initialized.

INPUTS
none

RESULTS

`ink_rect` rectangle filled with ink extents

`logical_rect`
rectangle filled with logical extents

33.10 playoutiter:GetLine

NAME
playoutiter:GetLine – get current line

SYNOPSIS
```
handle = playoutiter:GetLine()
```

FUNCTION
Gets the current line. The returned Pango layout line is owned by the the layout iterator and mustn't be freed.

Use the faster playoutiter:GetLineReadonly() if you do not plan to modify the contents of the line (glyphs, glyph widths, etc.).

INPUTS
none

RESULTS

`handle` the current line

33.11 playoutiter:GetLineExtents

NAME
playoutiter:GetLineExtents – get line extents

SYNOPSIS
```
ink_rect, logical_rect = playoutiter:GetLineExtents()
```

FUNCTION

Obtains the extents of the current line. This will return two tables that have the fields `x`, `y`, `width`, and `height` initialized.

Extents are in layout coordinates (origin is the top-left corner of the entire Pango layout). Thus the extents returned by this function will be the same width/height but not at the same x/y as the extents returned from playoutline:GetExtents().

INPUTS

none

RESULTS

`ink_rect` rectangle filled with ink extents

`logical_rect`
rectangle filled with logical extents

33.12 playoutiter:GetLineReadonly

NAME

playoutiter:GetLineReadonly – get line readonly

SYNOPSIS

```
handle = playoutiter:GetLineReadonly()
```

FUNCTION

Gets the current line for read-only access. The returned Pango layout line is owned by the the layout iterator and mustn't be freed.

This is a faster alternative to playoutiter:GetLine(), but the user is not expected to modify the contents of the line (glyphs, glyph widths, etc.).

INPUTS

none

RESULTS

`handle` the current line

33.13 playoutiter:GetLineYRange

NAME

playoutiter:GetLineYRange – get line yrange

SYNOPSIS

```
y0, y1 = playoutiter:GetLineYrange()
```

FUNCTION

Divides the vertical space in the Pango layout being iterated over between the lines in the layout, and returns the space belonging to the current line.

A line's range includes the line's logical extents. plus half of the spacing above and below the line, if playout:SetSpacing() has been called to set layout spacing. The Y positions are in layout coordinates (origin at top left of the entire layout).

INPUTS
none

RESULTS

`y0`	start of line
`y1`	end of line

33.14 playoutiter:GetRun

NAME
playoutiter:GetRun – get current run

SYNOPSIS
```
handle = playoutiter:GetRun()
```

FUNCTION
Gets the current run. The run is returned as a Pango glyph item object and it mustn't be freed.

When iterating by run, at the end of each line, there's a position with a `NULL` run, so this function can return a `NULL` handle. The `NULL` run at the end of each line ensures that all lines have at least one run, even lines consisting of only a newline. You can use pglyphitem:IsNull() to check if a glyph item is `NULL`.

Use the faster playoutiter:GetRunReadonly() if you do not plan to modify the contents of the run (glyphs, glyph widths, etc.).

INPUTS
none

RESULTS

`handle`	the current run as a Pango glyph item

33.15 playoutiter:GetRunExtents

NAME
playoutiter:GetRunExtents – get run extents

SYNOPSIS
```
ink_rect, logical_rect = playoutiter:GetRunExtents()
```

FUNCTION
Gets the extents of the current run in layout coordinates. This will return two tables that have the fields `x`, `y`, `width`, and `height` initialized.

Layout coordinates have the origin at the top left of the entire layout.

INPUTS
none

RESULTS

`ink_rect`	rectangle filled with ink extents

`logical_rect`
rectangle filled with logical extents

33.16 playoutiter:GetRunReadonly

NAME
playoutiter:GetRunReadonly – get current run for read-only access

SYNOPSIS
```
handle = playoutiter:GetRunReadonly()
```

FUNCTION
Gets the current run for read-only access. The run is returned as a Pango glyph item object and it mustn't be freed.

When iterating by run, at the end of each line, there's a position with a `NULL` run, so this function can return a `NULL` object. The `NULL` run at the end of each line ensures that all lines have at least one run, even lines consisting of only a newline. You can use pglyphitem:IsNull() to check if a glyph item is `NULL`.

This is a faster alternative to playoutiter:GetRun(), but the user is not expected to modify the contents of the run (glyphs, glyph widths, etc.).

INPUTS
none

RESULTS
`handle` the current run as a Pango glyph item

33.17 playoutiter:IsNull

NAME
playoutiter:IsNull – check if iterator is invalid

SYNOPSIS
```
bool = playoutiter:IsNull()
```

FUNCTION
Returns `True` if the iterator is `NULL`, i.e. invalid. If functions that allocate iterators fail, they might not throw an error but simply set the iterator to `NULL`. You can use this function to check if iterator allocation has failed in which case the iterator will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS
none

RESULTS
`bool` `True` if the iterator is `NULL`, otherwise `False`

33.18 playoutiter:NextChar

NAME
playoutiter:NextChar – move iterator to next character

SYNOPSIS
```
ok = playoutiter:NextChar()
```

FUNCTION
Moves the iterator forward to the next character in visual order.

This function returns whether motion was possible. If the iterator was already at the end of the layout, returns `False`.

INPUTS
none

RESULTS

`ok`	whether motion was possible

33.19 playoutiter:NextCluster

NAME
playoutiter:NextCluster – move iterator to next cluster

SYNOPSIS
```
ok = playoutiter:NextCluster()
```

FUNCTION
Moves the iterator forward to the next cluster in visual order.

This function returns whether motion was possible. If the iterator was already at the end of the layout, returns `False`.

INPUTS
none

RESULTS

`ok`	whether motion was possible

33.20 playoutiter:NextLine

NAME
playoutiter:NextLine – move iterator to next line

SYNOPSIS
```
ok = playoutiter:NextLine()
```

FUNCTION
Moves the iterator forward to the start of the next line.

This function returns whether motion was possible. If the iterator is already on the last line, returns `False`.

INPUTS

none

RESULTS

`ok` whether motion was possible

33.21 playoutiter:NextRun

NAME

playoutiter:NextRun – move iterator to next run

SYNOPSIS

```
ok = playoutiter:NextRun()
```

FUNCTION

Moves the iterator forward to the next run in visual order.

This function returns whether motion was possible. If the iterator was already at the end of the layout, returns `False`.

INPUTS

none

RESULTS

`ok` whether motion was possible

34 Pango layout line

34.1 playoutline:Free

NAME
playoutline:Free – free layout line

SYNOPSIS
```
playoutline:Free()
```

FUNCTION
Decrease the reference count of a Pango layout line by one.

If the result is zero, the line and all associated memory will be freed.

INPUTS
none

34.2 playoutline:GetExtents

NAME
playoutline:GetExtents – get extents of layout line

SYNOPSIS
```
ink_rect, logical_rect = playoutline:GetExtents()
```

FUNCTION
Computes the logical and ink extents of a layout line. This will return two tables that have the fields `x`, `y`, `width`, and `height` initialized.

See pfont:GetGlyphExtents() for details about the interpretation of the rectangles.

INPUTS
none

RESULTS

`ink_rect` extents of the glyph string as drawn

`logical_rect`
logical extents of the glyph string

34.3 playoutline:GetHeight

NAME
playoutline:GetHeight – get line height

SYNOPSIS
```
height = playoutline:GetHeight()
```

FUNCTION

Computes the height of the line, as the maximum of the heights of fonts used in this line.

Note that the actual baseline-to-baseline distance between lines of text is influenced by other factors, such as playout:SetSpacing() and playout:SetLineSpacing().

INPUTS

none

RESULTS

`height` return location for the line height

34.4 playoutline:GetLength

NAME

playoutline:GetLength – get line length

SYNOPSIS

```
len = playoutline:GetLength()
```

FUNCTION

Returns the length of the line, in bytes.

INPUTS

none

RESULTS

`len` the length of the line

34.5 playoutline:GetPixelExtents

NAME

playoutline:GetPixelExtents – get pixel extents

SYNOPSIS

```
ink_rect, logical_rect = playoutline:GetPixelExtents()
```

FUNCTION

Computes the logical and ink extents of `layout_line` in device units. This will return two tables that have the fields `x`, `y`, `width`, and `height` initialized.

This function just calls playoutline:GetExtents() followed by two pango.ExtentsToPixels() calls, rounding `ink_rect` and `logical_rect` such that the rounded rectangles fully contain the unrounded one (that is, passes them as first argument to pango.ExtentsToPixels()).

INPUTS

none

RESULTS

`ink_rect` extents of the glyph string as drawn

`logical_rect`
: logical extents of the glyph string

34.6 playoutline:GetRuns

NAME

playoutline:GetRuns – get runs in line

SYNOPSIS

```
table = playoutline:GetRuns()
```

FUNCTION

Gets the all runs in the layout line. The runs are returned inside a table where each table element is a Pango glyph item object that mustn't be freed.

INPUTS

none

RESULTS

`table` table containing all runs in the line as Pango glyph items

34.7 playoutline:GetXRanges

NAME

playoutline:GetXRanges – get x ranges

SYNOPSIS

```
ranges = playoutline:GetXRanges(start_index, end_index)
```

FUNCTION

Gets a list of visual ranges corresponding to a given logical range. This returns a list containing a number of tables that have the the fields `x1` and `x2` initialized to describe a range.

This list is not necessarily minimal - there may be consecutive ranges which are adjacent. The ranges will be sorted from left to right. The ranges are with respect to the left edge of the entire layout, not with respect to the line.

INPUTS

`start_index`
: start byte index of the logical range; if this value is less than the start index for the line, then the first range will extend all the way to the leading edge of the layout; otherwise, it will start at the leading edge of the first character

`end_index`
: ending byte index of the logical range; if this value is greater than the end index for the line, then the last range will extend all the way to the trailing edge of the layout; otherwise, it will end at the trailing edge of the last character

RESULTS

`ranges` a list containing a number of ranges

34.8 playoutline:IndexToX

NAME
playoutline:IndexToX – convert index to x position

SYNOPSIS
```
x_pos = playoutline:IndexToX(index, trailing)
```

FUNCTION
Converts an index within a line to a X position.

INPUTS

`index` byte offset of a grapheme within the layout

`trailing` an integer indicating the edge of the grapheme to retrieve the position of; if > 0, the trailing edge of the grapheme, if 0, the leading of the grapheme

RESULTS

`x_pos` x_offset in Pango units

34.9 playoutline:IsNull

NAME
playoutline:IsNull – check if layout line is invalid

SYNOPSIS
```
bool = playoutline:IsNull()
```

FUNCTION
Returns `True` if the layout line is `NULL`, i.e. invalid. If functions that allocate layout lines fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the object will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS
none

RESULTS

`bool` `True` if the layout line is `NULL`, otherwise `False`

34.10 playoutline:Reference

NAME
playoutline:Reference – increase reference count

SYNOPSIS
```
playoutline:Reference()
```

FUNCTION
Increase the reference count of a Pango layout line by one.

INPUTS
none

34.11 playoutline:XToIndex

NAME
playoutline:XToIndex – convert from x offset to index

SYNOPSIS
```
inside, index, trailing = playoutline:XToIndex(x_pos)
```

FUNCTION
Converts from x offset to the byte index of the corresponding character within the text of the layout.

If `x_pos` is outside the line, `index` and `trailing` will point to the very first or very last position in the line. This determination is based on the resolved direction of the paragraph; for example, if the resolved direction is right-to-left, then an X position to the right of the line (after it) results in 0 being stored in `index` and `trailing`. An X position to the left of the line results in `index` pointing to the (logical) last grapheme in the line and `trailing` being set to the number of characters in that grapheme. The reverse is true for a left-to-right line.

This function returns `False` if `x_pos` was outside the line, `True` if inside

INPUTS

`x_pos` the X offset (in Pango units) from the left edge of the line

RESULTS

`inside` `False` if `x_pos` was outside the line, `True` if inside

`index` calculated byte index for the grapheme in which the user clicked

`trailing` integer indicating where in the grapheme the user clicked; it will either be zero, or the number of characters in the grapheme; 0 represents the leading edge of the grapheme

35 Pango matrix

35.1 pmatrix:Concat

NAME
pmatrix:Concat – concat matrix

SYNOPSIS
`pmatrix:Concat(new_matrix)`

FUNCTION
Changes the transformation of the matrix to be the transformation given by first applying transformation given by `new_matrix` then applying the original transformation.

INPUTS

`new_matrix`
a Pango matrix

35.2 pmatrix:Get

NAME
pmatrix:Get – get matrix affine transformation

SYNOPSIS
`xx, xy, yx, yy, x0, y0 = pmatrix:Get()`

FUNCTION
Gets the affine transformation from the matrix and returns it.

INPUTS
none

RESULTS

`xx`	xx component of the affine transformation
`xy`	xy component of the affine transformation
`yx`	yx component of the affine transformation
`yy`	yy component of the affine transformation
`x0`	X translation component of the affine transformation
`y0`	Y translation component of the affine transformation

35.3 pmatrix:GetFontScaleFactor

NAME
pmatrix:GetFontScaleFactor – get font scale factor

SYNOPSIS
```
yscale = pmatrix:GetFontScaleFactor()
```

FUNCTION
Returns the scale factor of a matrix on the height of the font.

That is, the scale factor in the direction perpendicular to the vector that the X coordinate is mapped to. If the scale in the X coordinate is needed as well, use pmatrix:GetFontScaleFactors().

INPUTS
none

RESULTS

`yscale`	the scale factor of the matrix on the height of the font

35.4 pmatrix:GetFontScaleFactors

NAME
pmatrix:GetFontScaleFactors – get font scale factors

SYNOPSIS
```
xscale, yscale = pmatrix:GetFontScaleFactors()
```

FUNCTION
Calculates the scale factor of a matrix on the width and height of the font.

That is, `xscale` is the scale factor in the direction of the X coordinate, and `yscale` is the scale factor in the direction perpendicular to the vector that the X coordinate is mapped to.

Note that output numbers will always be non-negative.

INPUTS
none

RESULTS

`xscale`	scale factor in the x direction
`yscale`	scale factor perpendicular to the x direction

35.5 pmatrix:Init

NAME
pmatrix:Init – init matrix

SYNOPSIS
```
pmatrix:Init(xx, xy, yx, yy, x0, y0)
```

FUNCTION
Sets `matrix` to be the affine transformation given by `xx`, `xy`, `yx`, `yy`, `x0`, `y0`.

INPUTS

`xx`	xx component of the affine transformation
`xy`	xy component of the affine transformation
`yx`	yx component of the affine transformation
`yy`	yy component of the affine transformation
`x0`	X translation component of the affine transformation
`y0`	Y translation component of the affine transformation

35.6 pmatrix:InitIdentity

NAME
pmatrix:InitIdentity – init identity

SYNOPSIS
```
pmatrix:InitIdentity()
```

FUNCTION
Modifies the matrix to be an identity transformation.

INPUTS
none

35.7 pmatrix:Rotate

NAME
pmatrix:Rotate – rotate matrix

SYNOPSIS
```
pmatrix:Rotate(degrees)
```

FUNCTION
Changes the transformation represented by the matrix to be the transformation given by first rotating by `degrees` degrees counter-clockwise then applying the original transformation.

INPUTS

`degrees`	degrees to rotate counter-clockwise

35.8 pmatrix:Scale

NAME
pmatrix:Scale – scale matrix

SYNOPSIS
`pmatrix:Scale(scale_x, scale_y)`

FUNCTION
Changes the transformation represented by the matrix to be the transformation given by first scaling by `scale_x` in the X direction and `scale_y` in the Y direction then applying the original transformation.

INPUTS

`scale_x`	amount to scale by in X direction
`scale_y`	amount to scale by in Y direction

35.9 pmatrix:TransformDistance

NAME
pmatrix:TransformDistance – transform distance

SYNOPSIS
`tx, ty = pmatrix:TransformDistance(dx, dy)`

FUNCTION
Transforms the distance vector (`dx`,`dy`) by the matrix.

This is similar to pmatrix:TransformPoint(), except that the translation components of the transformation are ignored. The calculation of the returned vector is as follows:

```
dx2 = dx1 * xx + dy1 * xy;
dy2 = dx1 * yx + dy1 * yy;
```

Affine transformations are position invariant, so the same vector always transforms to the same vector. If (`x1`,`y1`) transforms to (`x2`,`y2`) then (`x1+dx1`,`y1+dy1`) will transform to (`x1+dx2`,`y1+dy2`) for all values of `x1` and `x2`.

INPUTS

`dx`	X component of a distance vector
`dy`	Y component of a distance vector

RESULTS

`tx`	transformed X component of a distance vector
`ty`	transformed Y component of a distance vector

35.10 pmatrix:TransformPixelRectangle

NAME

pmatrix:TransformPixelRectangle – transform pixel rectangle

SYNOPSIS

```
rc = pmatrix:TransformPixelRectangle(rect)
```

FUNCTION

First transforms the `rect` using the matrix, then calculates the bounding box of the transformed rectangle. The `rect` parameter must be a table that has the `x`, `y`, `width`, and `height` fields initialized. The return table will also have these fields initialized.

This function is useful for example when you want to draw a rotated Pango layout to an image buffer, and want to know how large the image should be and how much you should shift the layout when rendering.

For better accuracy, you should use pmatrix:TransformRectangle() on original rectangle in Pango units and convert to pixels afterward using pango.ExtentsToPixels()'s first argument.

INPUTS

`rect`	bounding box in device units

RESULTS

`rc`	transformed rectangle

35.11 pmatrix:TransformPoint

NAME

pmatrix:TransformPoint – transform point

SYNOPSIS

```
tx, ty = pmatrix:TransformPoint(x, y)
```

FUNCTION

Transforms the point (`x`, `y`) by the matrix.

INPUTS

`x`	X position
`y`	Y position

RESULTS

`tx`	transformed X position
`ty`	transformed Y position

35.12 pmatrix:TransformRectangle

NAME

pmatrix:TransformRectangle – transform rectangle

SYNOPSIS

```
rc = pmatrix:TransformRectangle(rect)
```

FUNCTION

First transforms `rect` using the matrix, then calculates the bounding box of the transformed rectangle. The `rect` parameter must be a table that has the `x`, `y`, `width`, and `height` fields initialized. The return table will also have these fields initialized.

This function is useful for example when you want to draw a rotated Pango layout to an image buffer, and want to know how large the image should be and how much you should shift the layout when rendering.

If you have a rectangle in device units (pixels), use pmatrix:TransformPixelRectangle().

If you have the rectangle in Pango units and want to convert to transformed pixel bounding box, it is more accurate to transform it first (using this function) and pass the result to pango.ExtentsToPixels(), first argument, for an inclusive rounded rectangle. However, there are valid reasons that you may want to convert to pixels first and then transform, for example when the transformed coordinates may overflow in Pango units (large matrix translation for example).

INPUTS

`rect`	bounding box in Pango units

RESULTS

`rc`	transformed rectangle

35.13 pmatrix:Translate

NAME

pmatrix:Translate – translate matrix

SYNOPSIS

```
pmatrix:Translate(tx, ty)
```

FUNCTION

Changes the transformation represented by the matrix to be the transformation given by first translating by (`tx`, `ty`) then applying the original transformation.

INPUTS

`tx`	amount to translate in the X direction
`ty`	amount to translate in the Y direction

36 Pango tab array

36.1 ptabarray:Copy

NAME
ptabarray:Copy – copy tab array

SYNOPSIS
```
tabs = ptabarray:Copy()
```

FUNCTION
Copies a Pango tab array.

This function returns the newly allocated Pango tab array, which should be freed with ptabarray:Free().

INPUTS
none

RESULTS

`tabs`	the newly allocated Pango tab array

36.2 ptabarray:Free

NAME
ptabarray:Free – free tab array

SYNOPSIS
```
ptabarray:Free()
```

FUNCTION
Frees a tab array and associated resources.

INPUTS
none

36.3 ptabarray:GetPositionsInPixels

NAME
ptabarray:GetPositionsInPixels – get positions in pixels flag

SYNOPSIS
```
ok = ptabarray:GetPositionsInPixels()
```

FUNCTION
Returns `True` if the tab positions are in pixels, `False` if they are in Pango units.

INPUTS
none

RESULTS

`ok`	whether positions are in pixels

36.4 ptabarray:GetSize

NAME
ptabarray:GetSize – get number of tab stops

SYNOPSIS
```
n = ptabarray:GetSize()
```

FUNCTION
Gets the number of tab stops in the tab array.

INPUTS
none

RESULTS

`n`	the number of tab stops in the array

36.5 ptabarray:GetTab

NAME
ptabarray:GetTab – get tab

SYNOPSIS
```
alignment, location = ptabarray:GetTab(tab_index)
```

FUNCTION
Gets the alignment and position of a tab stop.

INPUTS

`tab_index`	tab stop index

RESULTS

`alignment`	tab stop alignment
`location`	tab stop position

36.6 ptabarray:GetTabs

NAME
ptabarray:GetTabs – get tabs

SYNOPSIS
```
tabs = ptabarray:GetTabs()
```

FUNCTION
Returns a table containing alignments and locations of all tab stops. The returned table contains a subtable for each tab stop. Each subtable has the fields `Align` and `Pos` initialized to the respective tab stop alignment and position.

INPUTS
none

RESULTS

`tabs` table containing alignments and locations of all tab stops

36.7 ptabarray:IsNull

NAME
ptabarray:IsNull – check if tab array is invalid

SYNOPSIS
```
bool = ptabarray:IsNull()
```

FUNCTION
Returns `True` if the tab array is `NULL`, i.e. invalid. If functions that allocate objects fail, they might not throw an error but simply set the object to `NULL`. You can use this function to check if object allocation has failed in which case the tab array will be `NULL`.

Also, certain getter functions can return a `NULL` object in case the object doesn't exist. You can use this function to check if a getter function returned a `NULL` handle.

INPUTS
none

RESULTS

`bool` `True` if the tab array is `NULL`, otherwise `False`

36.8 ptabarray:Resize

NAME
ptabarray:Resize – resize tab array

SYNOPSIS
```
ptabarray:Resize(new_size)
```

FUNCTION
Resizes a tab array to the size specified by `new_size`.

You must subsequently initialize any tabs that were added as a result of growing the array.

INPUTS

`new_size` new size of the array

36.9 ptabarray:SetTab

NAME

ptabarray:SetTab – set tab stop

SYNOPSIS

```
ptabarray:SetTab(tab_index, alignment, location)
```

FUNCTION

Sets the alignment and location of a tab stop. See pango.TabArrayWithPositions for a list of supported alignments for the `alignment` parameter.

INPUTS

`tab_index`
: the index of a tab stop

`alignment`
: tab alignment

`location` tab location in Pango units

Appendix A Licenses

A.1 LGPL license

GNU LESSER GENERAL PUBLIC LICENSE Version 2.1, February 1999

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent

license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or

any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the

Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

A.2 HarfBuzz license

HarfBuzz is licensed under the so-called "Old MIT" license. Details follow. For parts of HarfBuzz that are licensed under different licenses see individual files names COPYING in subdirectories where applicable.

A.3 Expat license

distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.4 Fontconfig license

Copyright (C) 2000,2001,2002,2003,2004,2006,2007 Keith Packard
Copyright 2005 Patrick Lam
Copyright 2007 Dwayne Bailey and Translate.org.za
Copyright 2009 Roozbeh Pournader
Copyright 2008,2009,2010,2011,2012,2013,2014,2015,2016,2017,2018,2019,2020 Red Hat, Inc.
Copyright 2008 Danilo Aegan
Copyright 2012 Google, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of the author(s) not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. The authors make no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

THE AUTHOR(S) DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL THE AUTHOR(S) BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

A.5 Pixman license

The following is the MIT license, agreed upon by most contributors. Copyright holders of new code should use this license statement where possible. They may also add themselves to the list below.

Copyright 1987, 1988, 1989, 1998 The Open Group
Copyright 1987, 1988, 1989 Digital Equipment Corporation
Copyright 1999, 2004, 2008 Keith Packard

A.6 Libxml2 license

BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Index

C

P